ELECTRICAL ENGINEERING & COMPUTER SCIENCE

EECS 373: Intro to Embedded System Design

Lecture 9: “Timers”

Dr. Alanson Sample
University of Michigan

Outline for Today's Lecture

Timers

- Background
- Anatomy of a Timer
- Capture & Compare
- Pulse Width Modulation

SysTick Timers

Virtual Timers

Reading

- STM32L4+ Application Developers Reference Manual (RM0432)
  - Section 37: Advanced-control timers (TIM1/TIM8
  - Section 38: General-purpose timers (TIM2/TIM3/TIM4/TIM5)

- Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C (third Edition) -Dr. Yifeng Zhu
  - Chapter 15 General Purpose Timers

STM32 Youtube Tutorial

- https://www.youtube.com/watch?v=VfbW6nfG4kw&list=PLEBQazB0HUyRYuzfi4cIXsKUSgorErmBv&index=6

Background on Timers & Clocks

iPhone Clock App

- World Clock – display real time in multiple time zones
- Alarm – alarm at certain (later) time(s).
- Stopwatch – measure elapsed time of an event
- Timer – count down time and notify when count becomes zero

Motor/Light Control

- Servo motors – PWM signal provides control signal
- DC motors – PWM signals control power delivery
- RGB LEDs – PWM signals allow dimming through current-mode control

Timing functions in computer systems

Periodically interrupt CPU to perform tasks
- Sample sensor readings (temperature, pressure, etc.)
- Generate music samples

Provide accurate time delays
- Instead of software loops

Generate pulses or periodic waveforms
- PWM signal for motor control
- Strobe pulse for an external device

Measure duration of an external event
- Tachometer signal period to measure motor speed

Measure the passage of “user time”
- Date: Month, Day, Year
- Time: HH:MM:SS:mmm

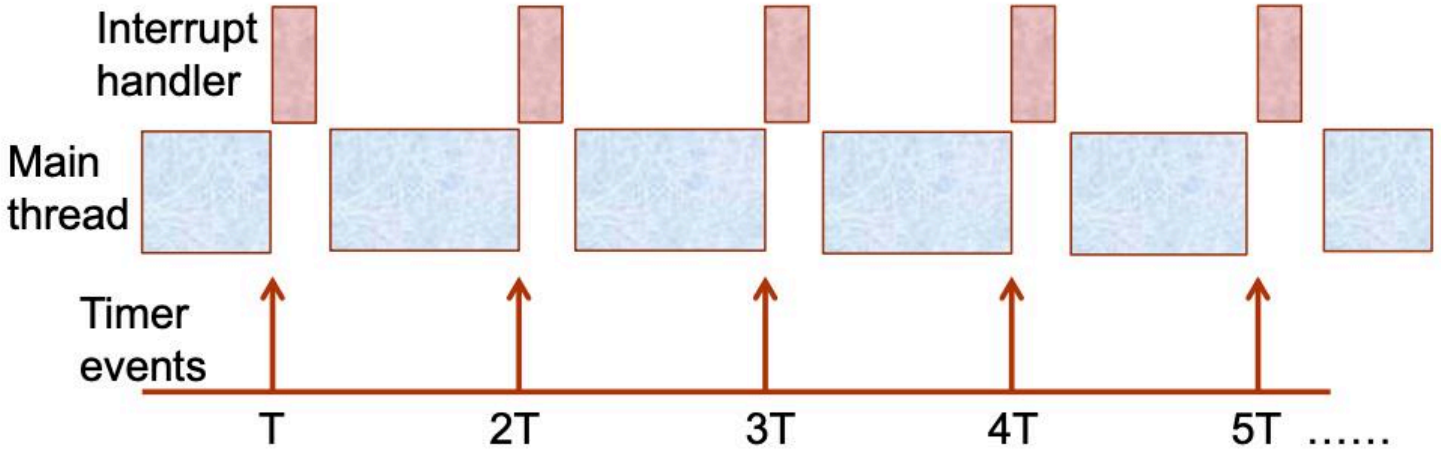
Measuring time on an MCU

Timer module interrupts the main thread every T sec

- Timer period is usually programmable

Interrupt handler performs required operations

- Operations usually include clearing a flag in the timer



How do electronics detect the passage of time?

How do humans get a sense of time?

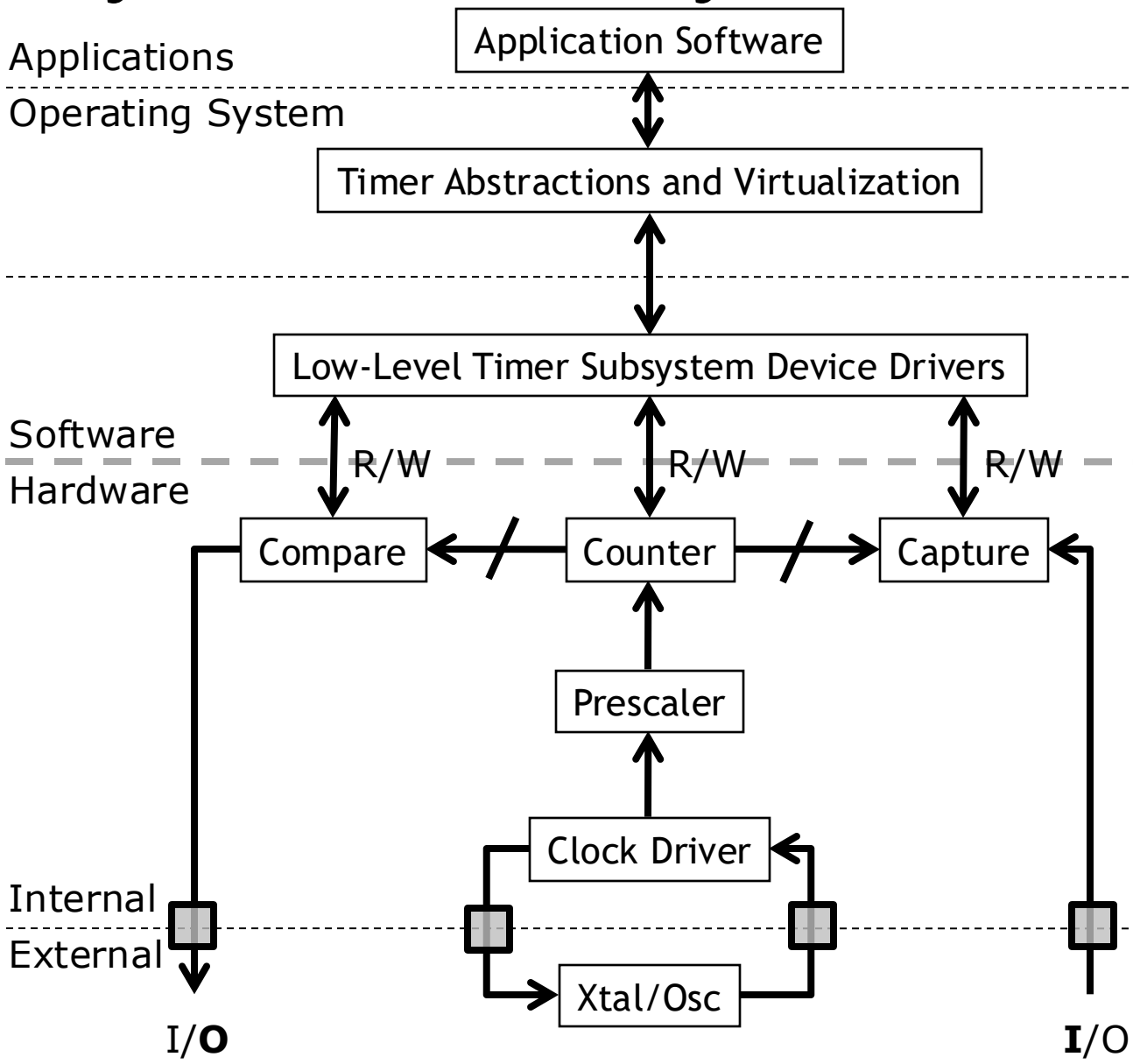
- From the natural world
- For humans the rise and fall of the Sun
- Then we made clocks: Spring / Mass time constant

Electronics devices need some physical principal to mark the passage of time. What do they use?

>> 60 second brainstorm talk with your neighbor <<

Anatomy of a Timer

Anatomy of a timer system



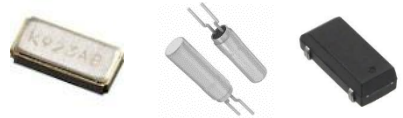

```
...
timer_t timerX;
initTimer();
...
startTimerOneShot(timerX, 1024);
...
stopTimer(timerX);
```

```
typedef struct timer {
  timer_handler_t handler;
  uint32_t time;
  uint8_t mode;
  timer_t* next_timer;
} timer_t;
```

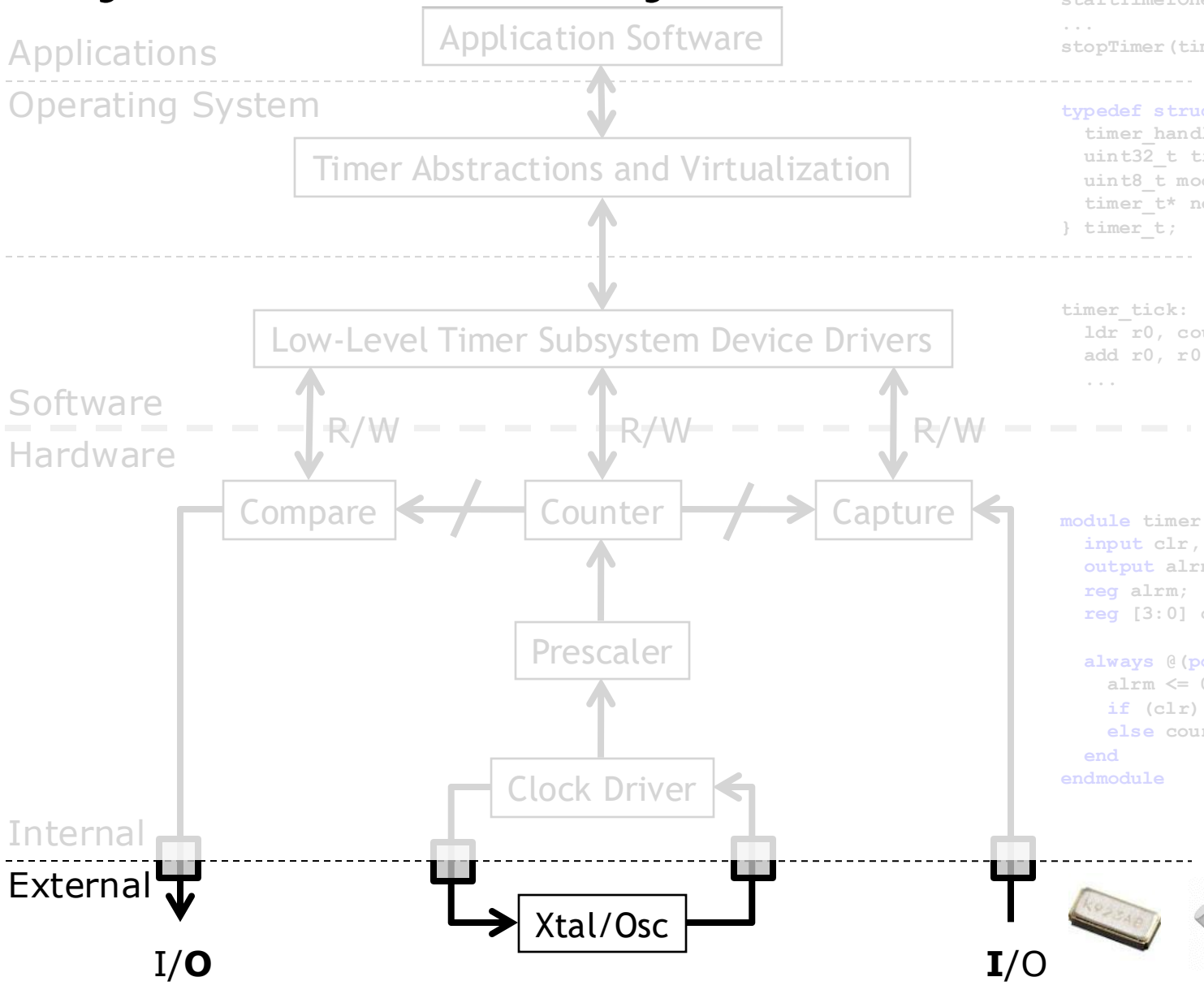
```
timer_tick:
  ldr r0, count;
  add r0, r0, #1
  ...
```

```
module timer(clr, ena, clk, alrm);
  input clr, ena, clk;
  output alrm;
  reg alrm;
  reg [3:0] count;

  always @(posedge clk) begin
    alrm <= 0;
    if (clr) count <= 0;
    else count <= count+1;
  end
endmodule
```



Anatomy of a timer system



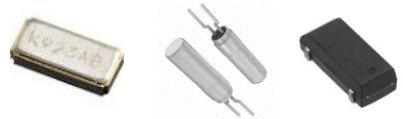

```
timer_t timerX;
initTimer();
...
startTimerOneShot(timerX, 1024);
...
stopTimer(timerX);
```

```
typedef struct timer {
  timer_handler_t handler;
  uint32_t time;
  uint8_t mode;
  timer_t* next_timer;
} timer_t;
```

```
timer_tick:
  ldr r0, count;
  add r0, r0, #1
  ...
```

```
module timer(clr, ena, clk, alrm);
  input clr, ena, clk;
  output alrm;
  reg alrm;
  reg [3:0] count;

  always @(posedge clk) begin
    alrm <= 0;
    if (clr) count <= 0;
    else count <= count+1;
  end
endmodule
```



Oscillators – Crystal

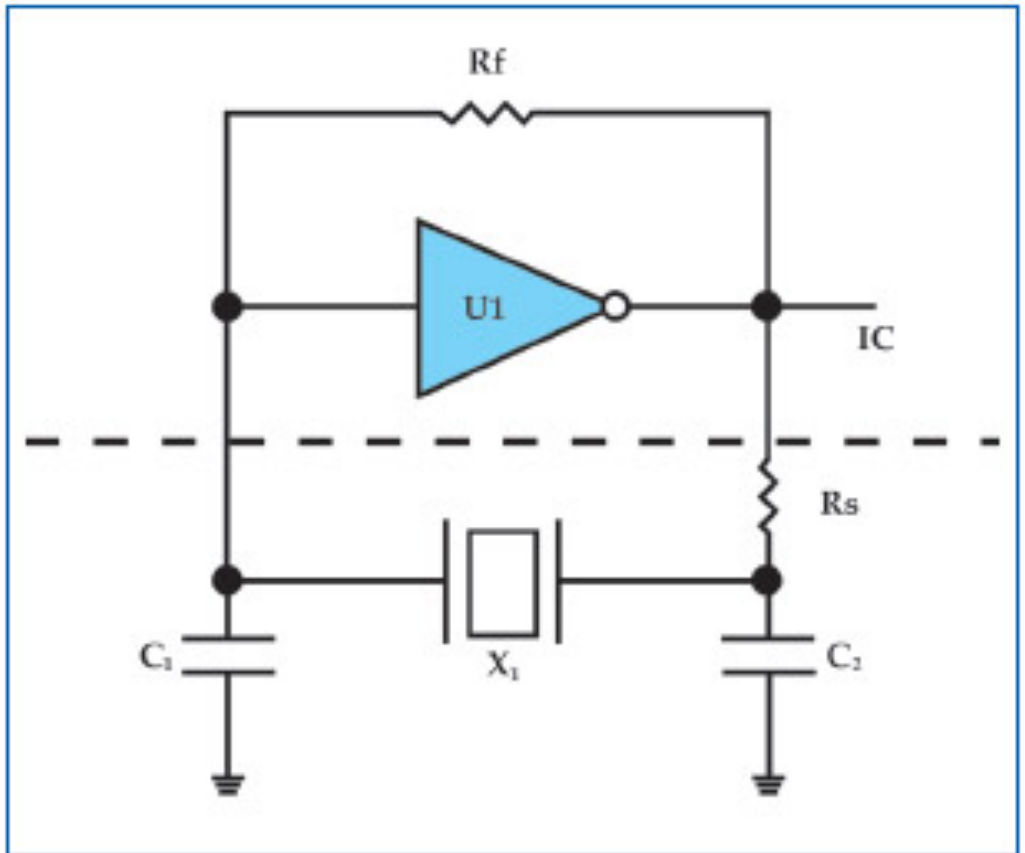
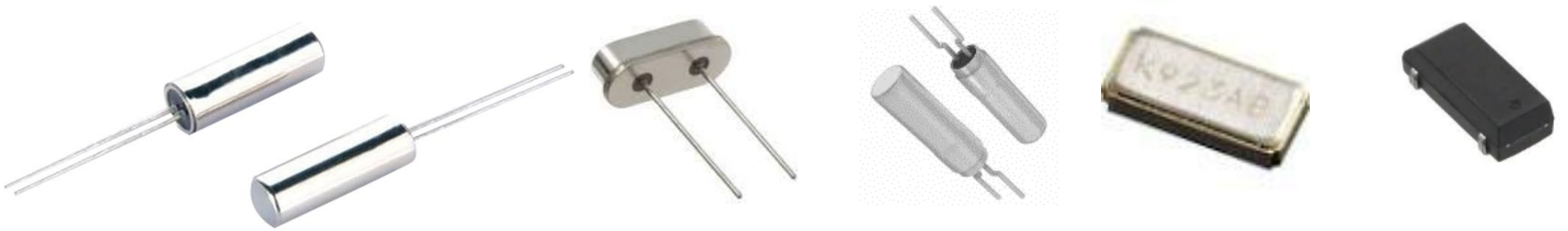
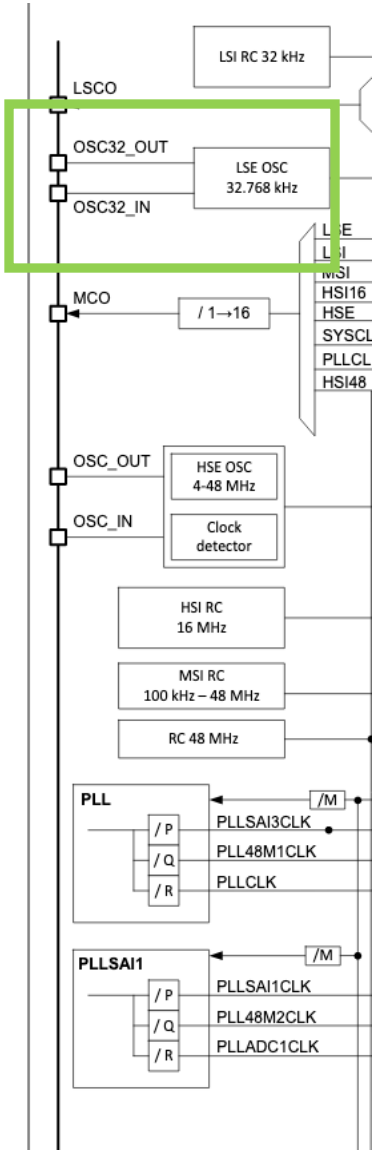


Figure 1: Fundamental Mode Isolated Pierce-Gate Oscillator

Types of Clock Sources : Crystals

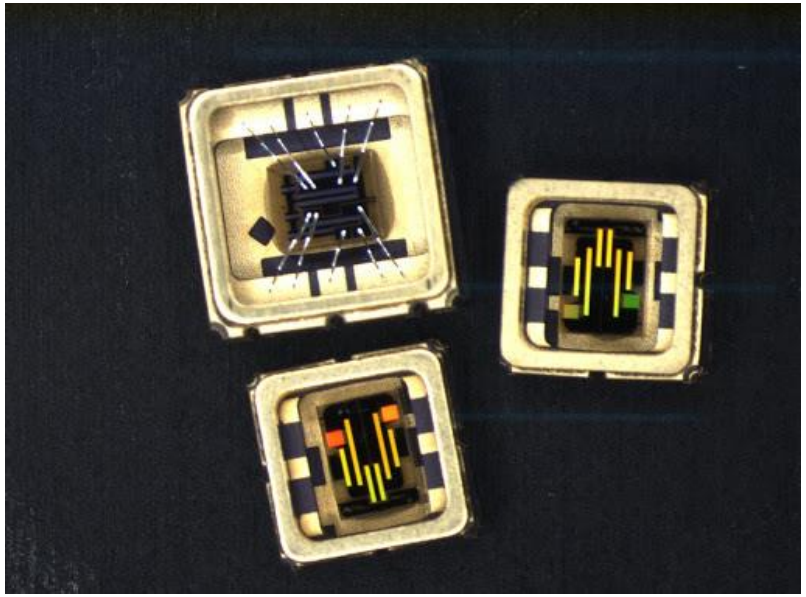
- Low frequency: Typically 32.768 KHz
- Built with registers for
  - Years, Months, Days, Hours, Mins, Seconds
- Alarms: hour, min, day
- Accessed via Memory-mapped I/O

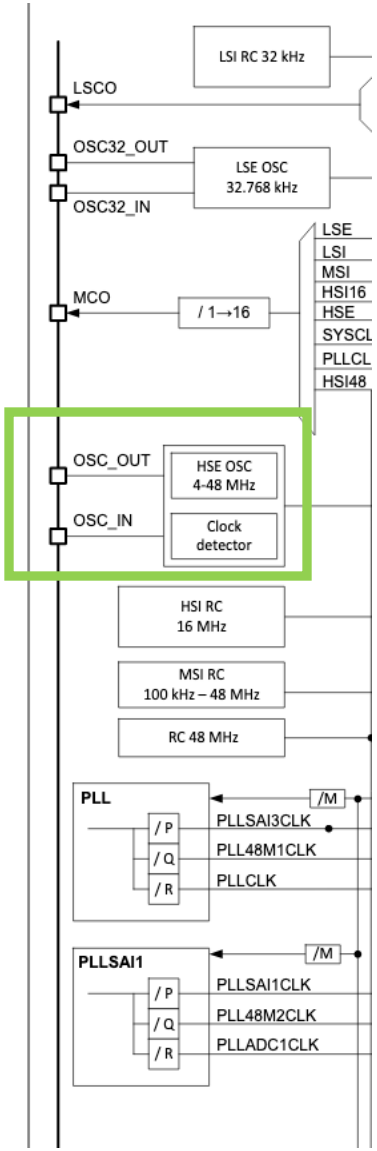




Types of Clock Sources: SAW Resonators

- High frequency clock sources
  - External crystals or ceramic resonators (SAW)
  - Typically: 10MHz -100MHz
  - Very accurate
  - Limited number of frequencies



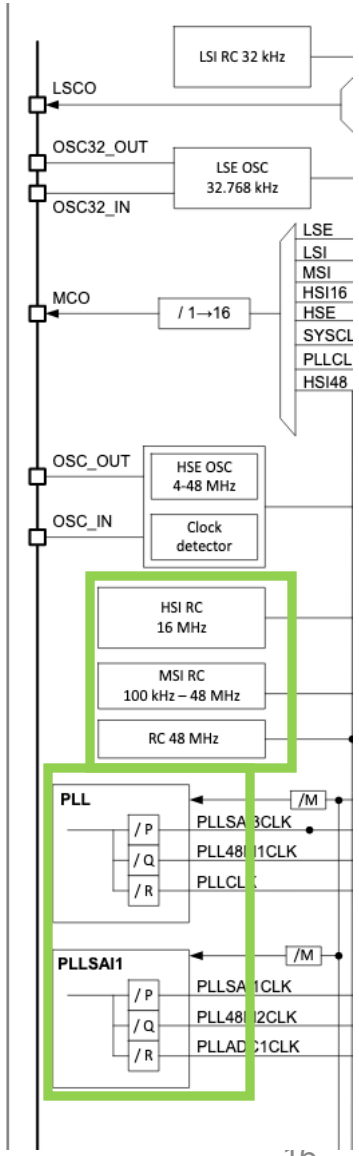


Internal clock sources

- Digitally controlled Oscillators DCO
- 100kHz to ~10MHz
- Less accuracy
- Lots of frequencies / very agile

- STM32L4 has a AHB and APB clock speed of up to 120MHz.
  … but max external and internal oscillator frequency is 48MHz

- PLL – Phase Lock Loop
  - “multiple” in put frequency to create faster clock
  - higher clock noise and power

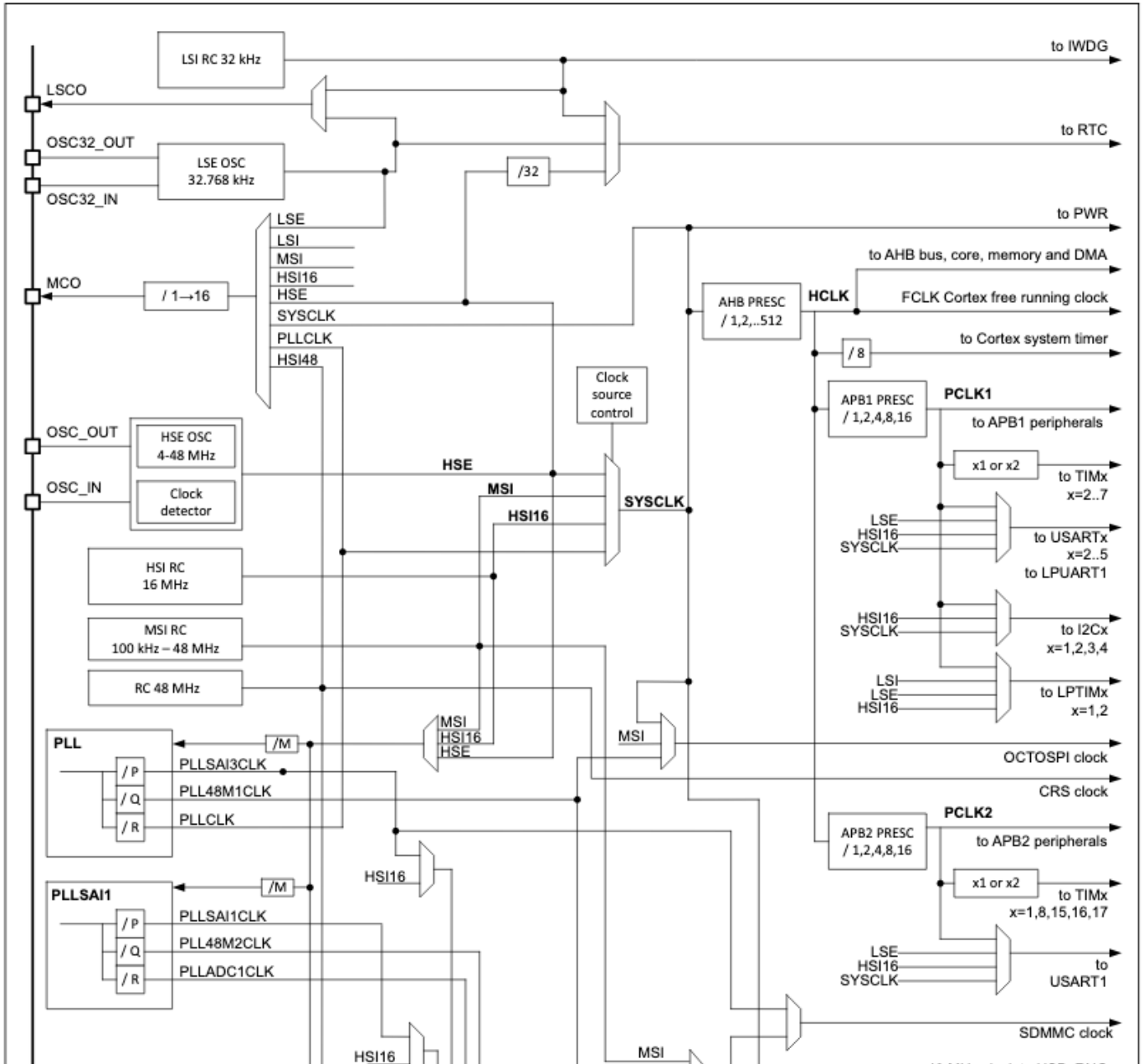


Clock Distribution

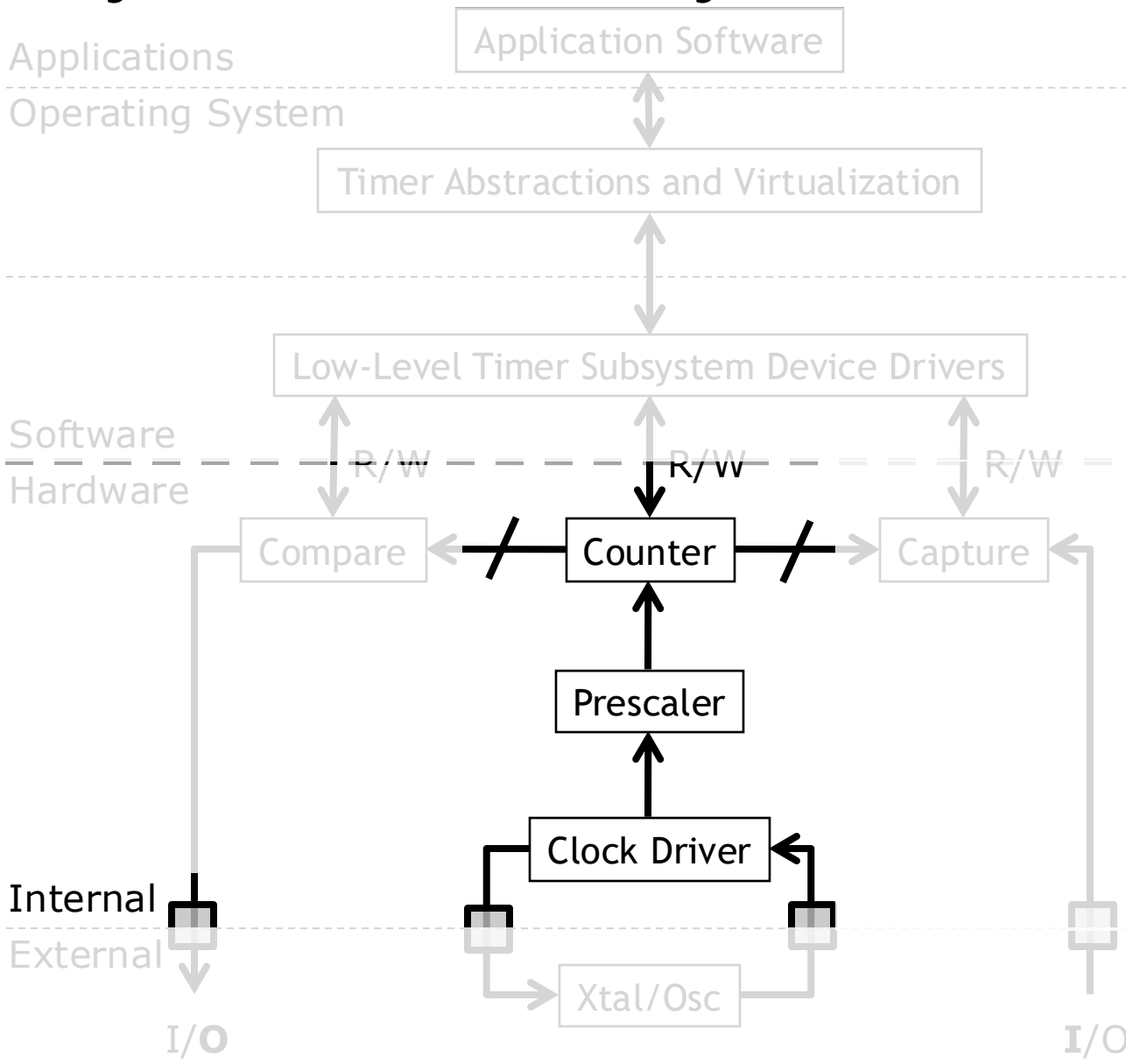
Good to be aware of

- Multiple clock sources
- Prescalers / clock dividers
- Flexible clock disturbing
- Send different frequency to different peripheral
  - Core, APB, Timers, UART, etc. can all have different clocks based on the applications need

Figure 16. Clock tree for STM32L4Rxxx and STM32L4Sxxx devices



Anatomy of a timer system




```
timer_t timerX;
initTimer();
...
startTimerOneShot(timerX, 1024);
...
stopTimer(timerX);
```

```
typedef struct timer {
  timer_handler_t handler;
  uint32_t time;
  uint8_t mode;
  timer_t* next_timer;
} timer_t;
```

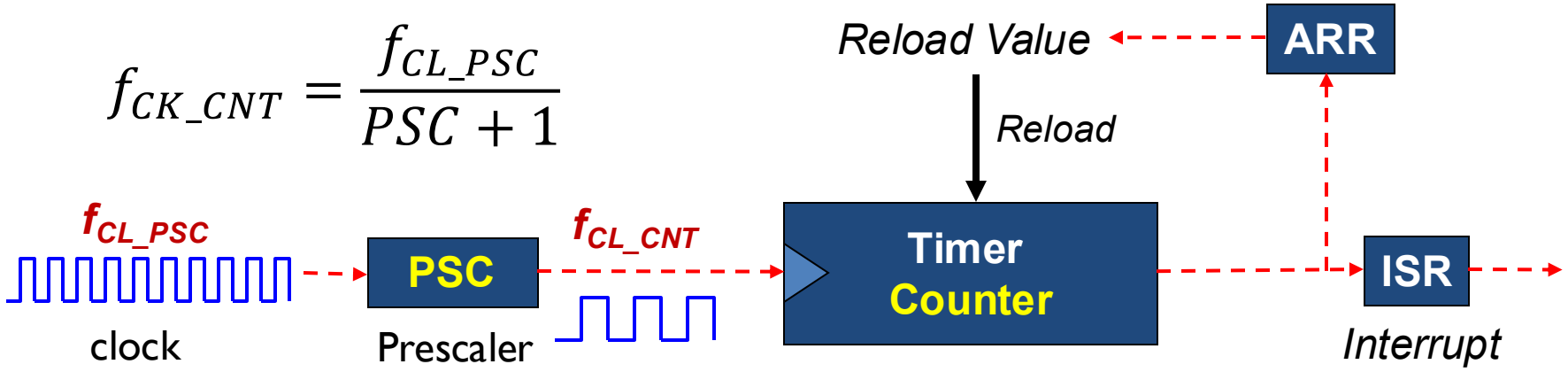
```
timer_tick:
  ldr r0, count;
  add r0, r0, #1
  ...
```

```
module timer(clr, ena, clk, alrm);
  input clr, ena, clk;
  output alrm;
  reg alrm;
  reg [3:0] count;

  always @(posedge clk) begin
    alrm <= 0;
    if (clr) count <= 0;
    else count <= count+1;
  end
endmodule
```

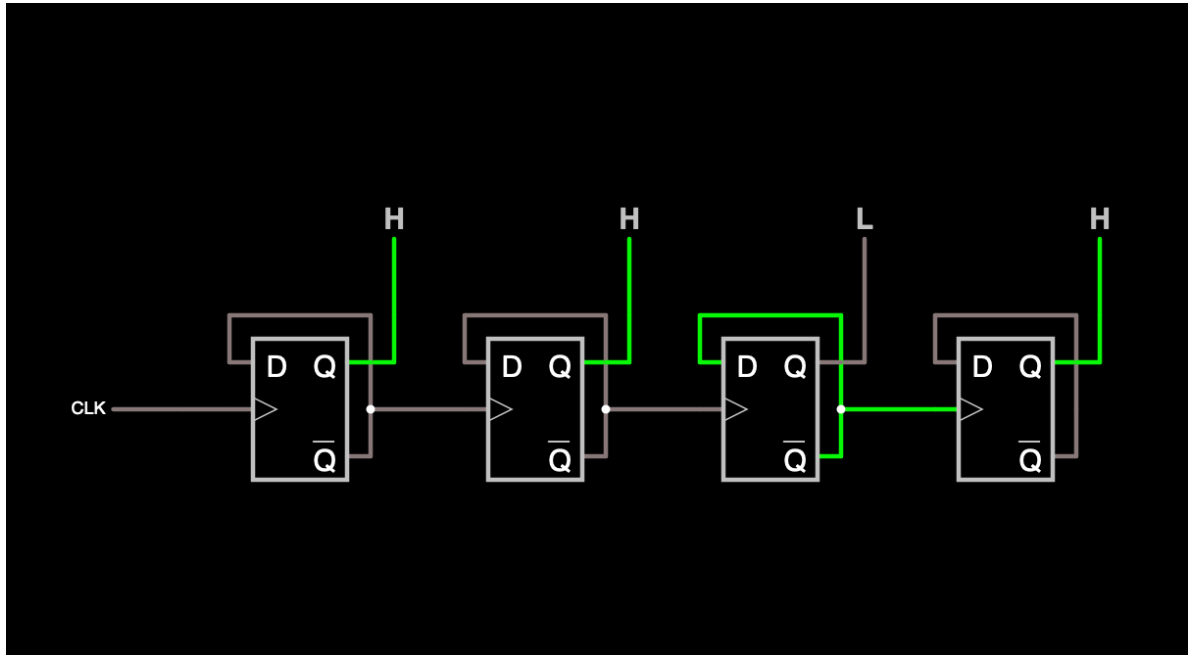
Counter Basics

$$f_{CK\_CNT} = \frac{f_{CL\_PSC}}{PSC + 1}$$



Falstad Clock Divider

https://tinyurl.com/274tn4b6



https://www.falstad.com/circuit/circuitjs.html?ctz=CQAgjCAMB0l3BWK0ECYCcAWT7GUwOxwDMICkZZkAUGAkmAGzhioiOYtsULUBKIALRgCzMNiGpMFcZwhswcEKhTLVFGLwDuXcBLCsQADg3UdBtifDMrNczZljDNALLhUR4zI-Woa7boW7p5BdsGBPrZmgfqGIsxhgpghEkkholDRgsQKhtmWpjppuvm6rkI5XhUKCf5ZsIXCsdxZyY0NUUVtDcVRdEhNcnnoTtxk0Ux6nMUNifFTQg2zExlLcRJhqAQKEIsKGZtGIYYenp3K27qnZdRu11bXkxQqAfNW86HRe1VvhRc7nG+fXo4FWJykZUy-RAxGIoxhRiGYwCmEqDVRBUyOgxVVhDixMLhuKJnzc2DOFHJvmeKGiOKCVM+2O6hip53pEnpB2ieIW32WAHtwOxOJTIOhPEgYBBnsLiNQhYYOH5pBLKNK-GAFZRIWK1VLoDLwNrUKRdSBVZLkEatULKubLerDZrqEA

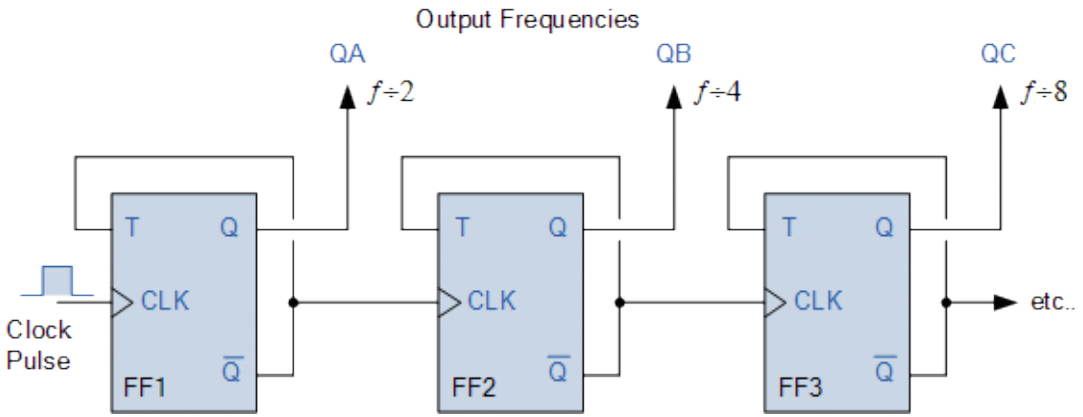
Clock Dividers vs Prescaler

Clock Dividers on STM32

- Divides the clock frequency by powers of 2
- Simple device: uses a few Flip Flops
- Fixed values: /2 /4 /8

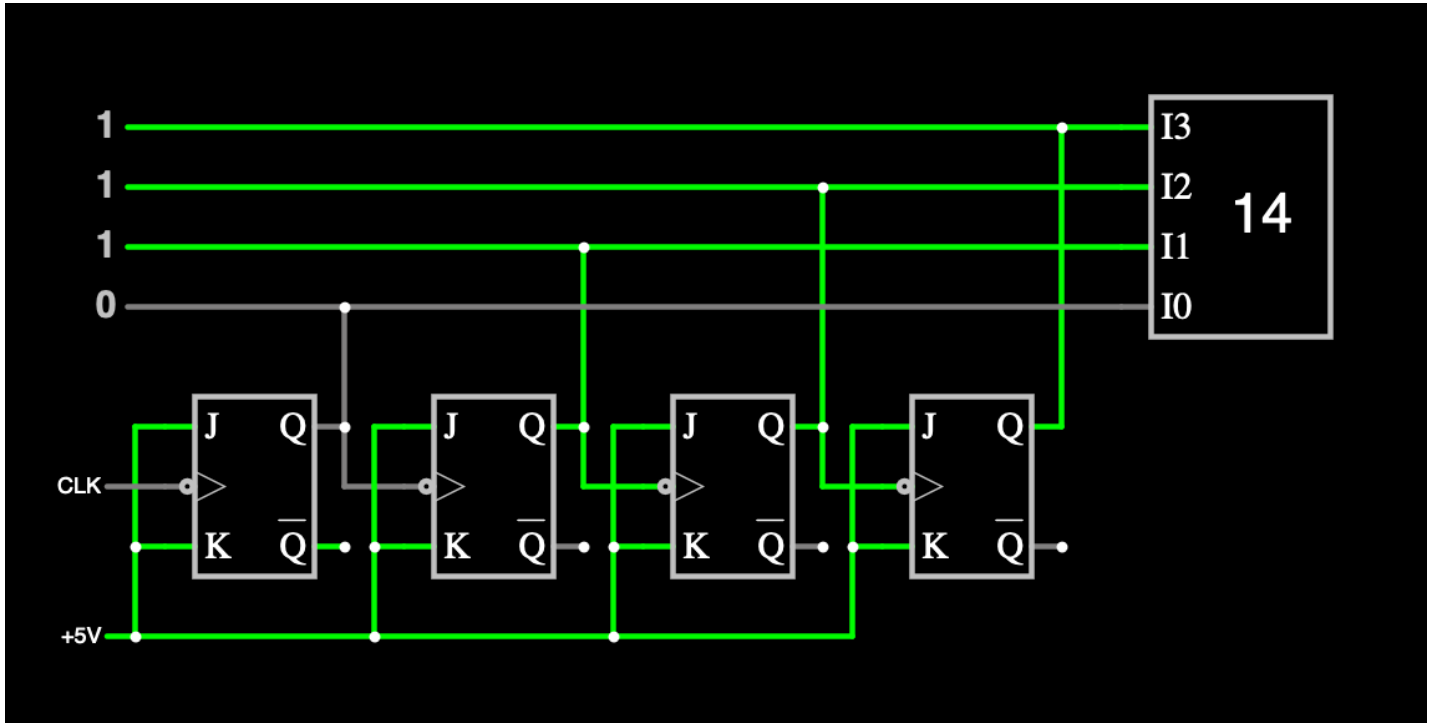
Prescalers on STM32

- Divide the counter clock frequency by any factor between 1 and 65536
- Based on a 16-bit counter
- Controlled through a 16-bit register (in the TIMx_PSC register)



Falstad Counter

https://tinyurl.com/yjtmtmro



https://www.falstad.com/circuit/circuitjs.html?ctz=CQAgjCAMB0l3BWcMBMcUHYMGZIA4UA2ATmIxAUgoqoQFMBaMMAKDAUJDypRQBZwKPCF4DabDiPwj+Ivj1lVIEztmydRIdcM3j2nPoQGa+3GWIosA7uAwbZYOyMnKbjjZLQ6X1rZGOyuMY+NkHOqurhUL582CjmILHxKCGJcVGGwZyuiUYJfKYJOQU6siUieMLFhULCmRVVMTWViYVq2U3C7VpoWpE5cVTdgw3Rob21fRotAxMtI7q+C4GRizZeCWDEyYq+WzsC+6M5R5Mbkyfbo0fdl-HDvbe+Zpp5ayBvsm8zvt-CLz8bH8PgInkDQZEbv0WAAILg8STpFKcCDJOAiaBIFCYqBQTGwkFTHpEpSJWi4pT4mwcBQCGmE4pIsrpYgdUL+BJhMBCMa2eyHJxgQhKFgAWQowsJhTeGIQYrS8VZiWESuS+PFXJ5hW5OIl8vc4El2slarlfC2FCueSMVDyVD4vnpeQQVrEMRZnBdirZfkOPK9gkabkFkoDQpFAHstEg7YkwJB4kgYPBIGRCBwUrjWFHIrHzQnqMn4GmM0nwCwc+Q8-HE3iUyXCJmqNmtMJqwWk7Bi3ZS1mWEA

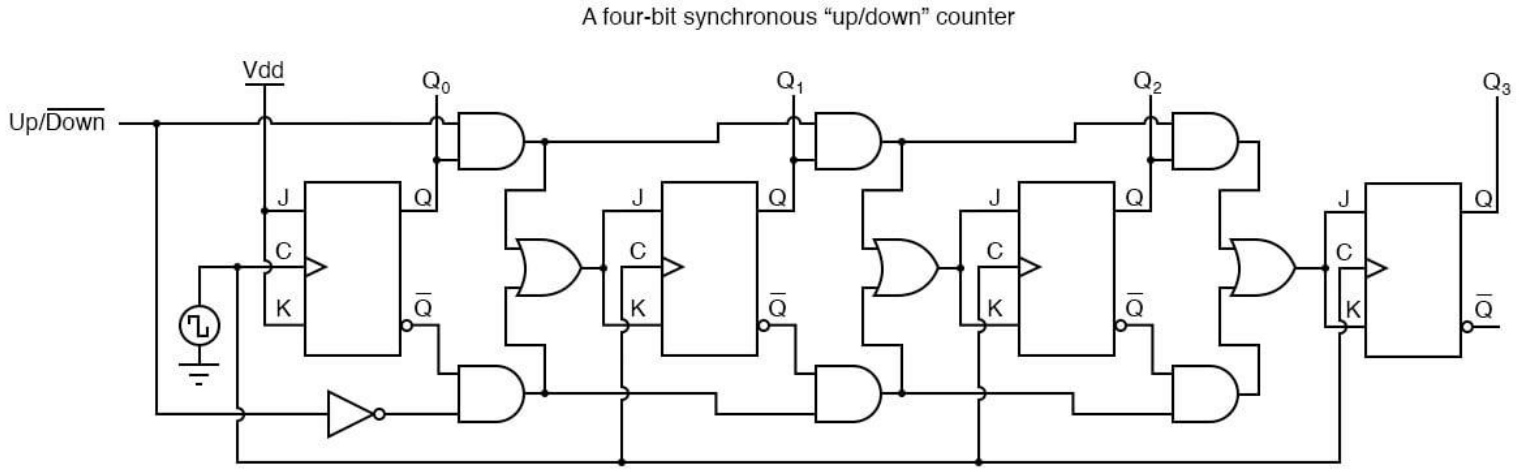
Counters

Different Modes
- Count up
- Count down
- Hold

Read and Write Data
- Load data into DFFs
- Read data out of DFFs

Number of bits
- 4, 8, 16, 32, 64

JK Flip Flop Up Down counter



The topology of the counters not important for EECS373, but its behavior is

Capture & Compare

Two primary timer modes

Measure how long something takes

- “Capture” how long something takes to happen

Have something happen once or every X time period

- “Compare” a clock source to a desired time and act

Example # 1: Capture

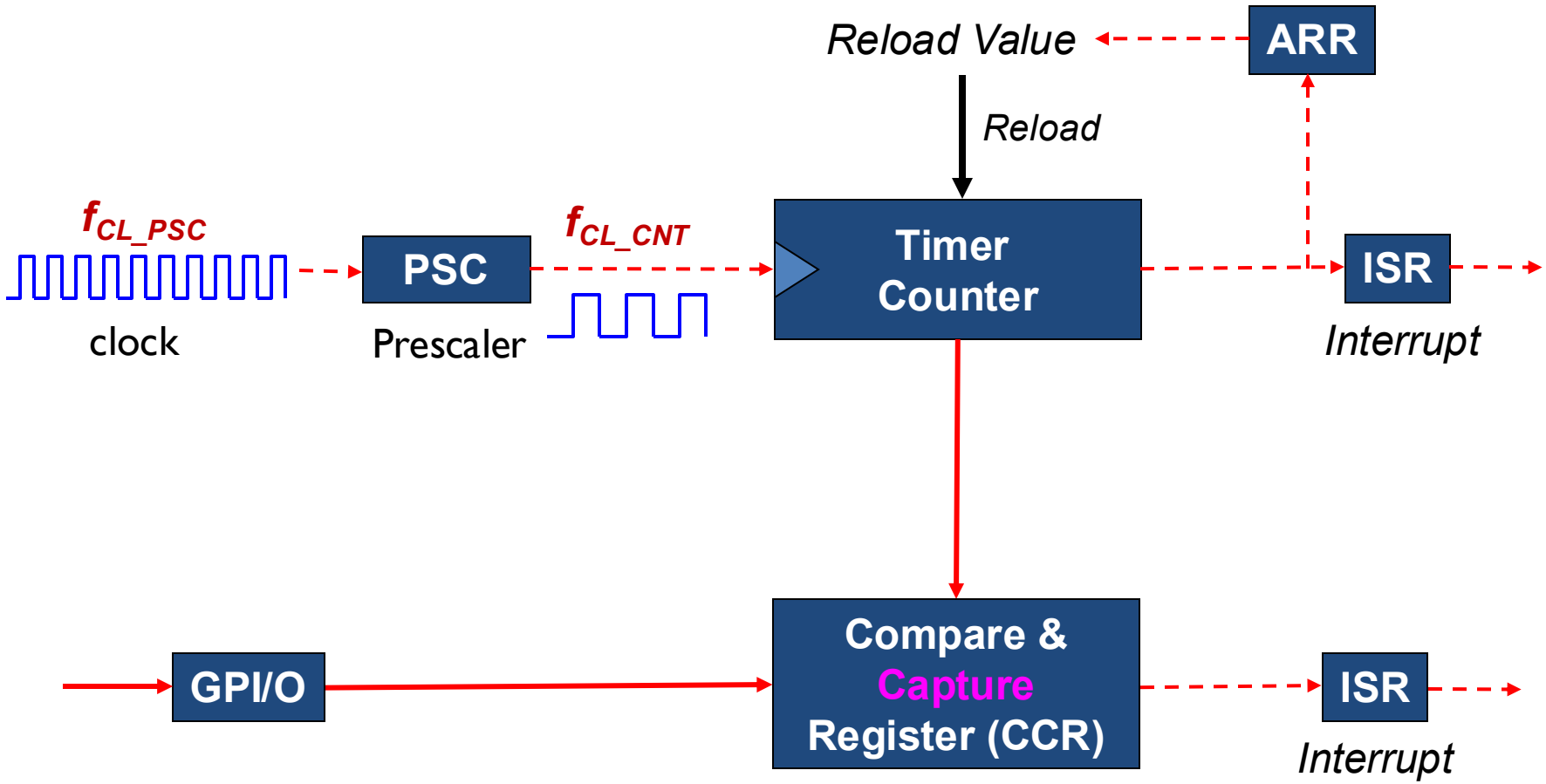
Say you have a fan and you want to know how fast it is spinning. One way to do that is to have it throw a *GPIO* interrupt every time it completes a rotation.

Right idea, but might take a while to process the interrupt, heavily loaded system might see slower fan than actually exists.

Solution:

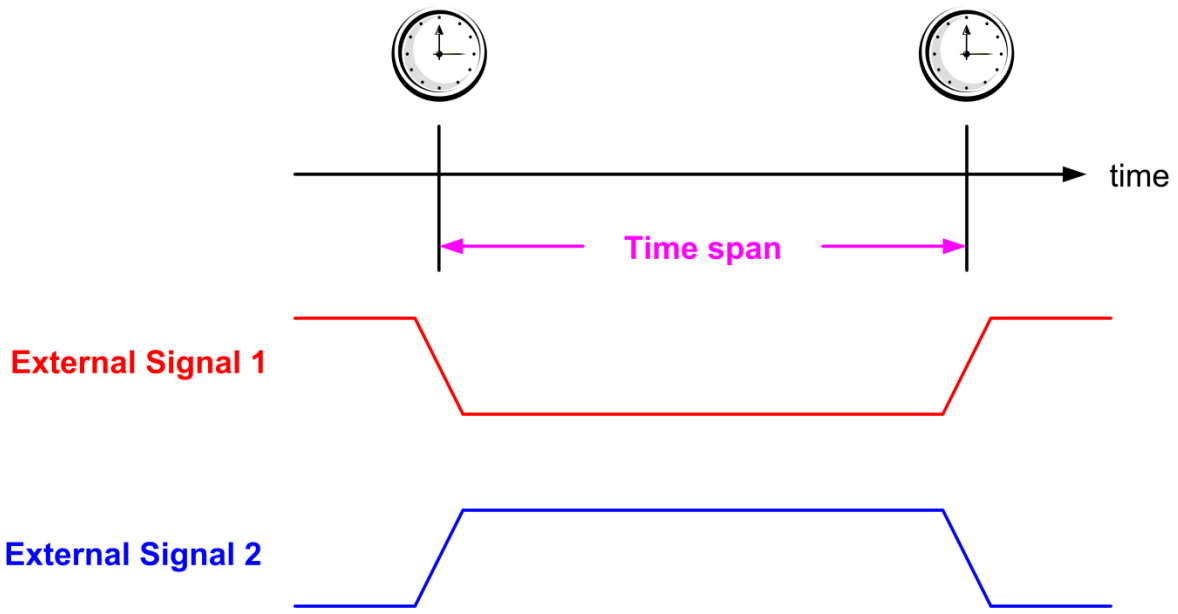
Have the timer note *immediately* how long it took and then generate the interrupt. Also restart timer immediately.

Timer: Input Capture



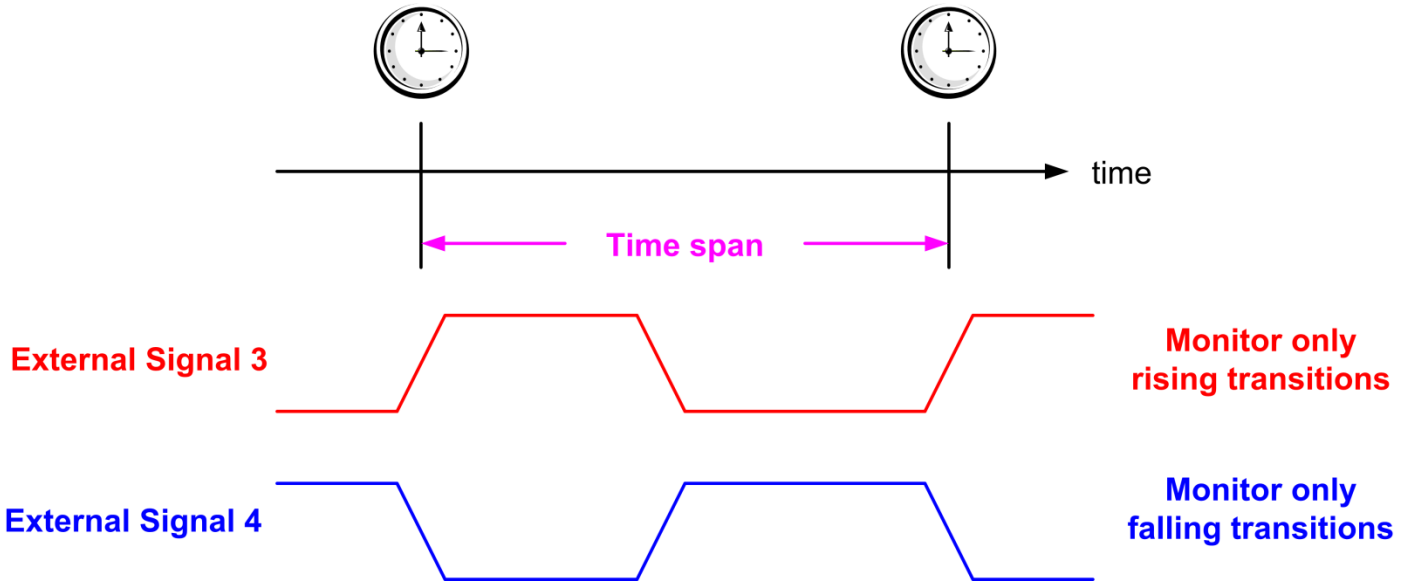
Input Capture

Monitor both rising and falling edge

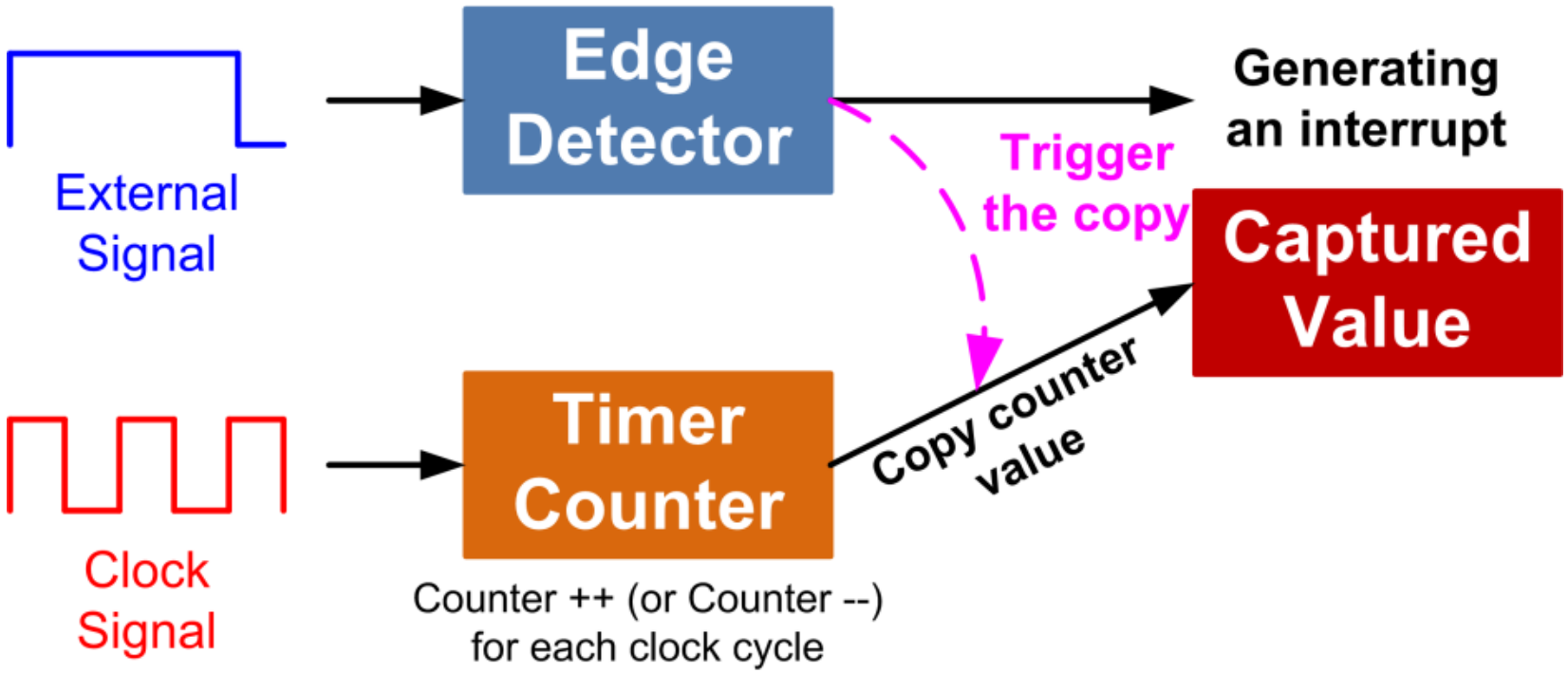


Input Capture

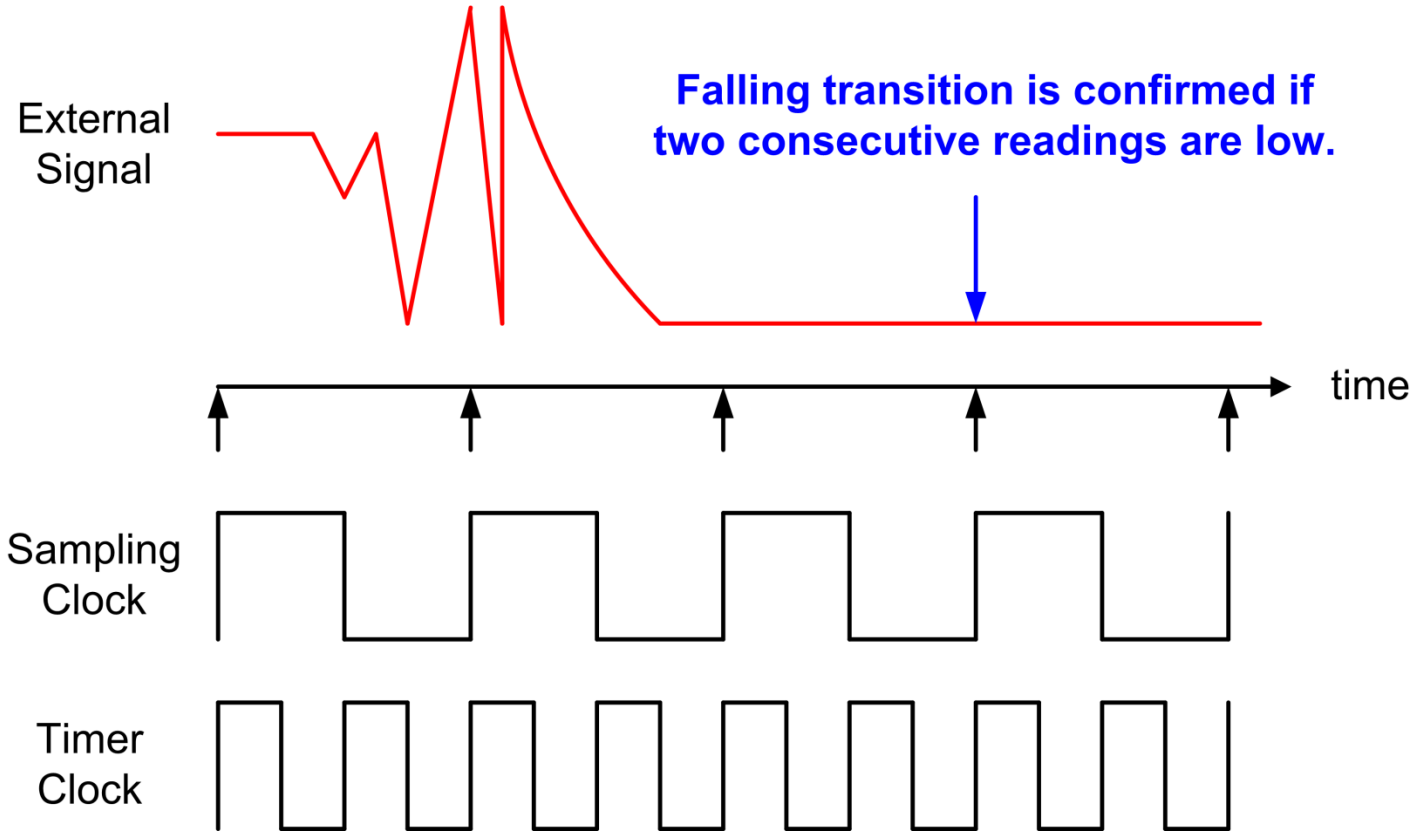
Monitor only rising edges or only falling edge



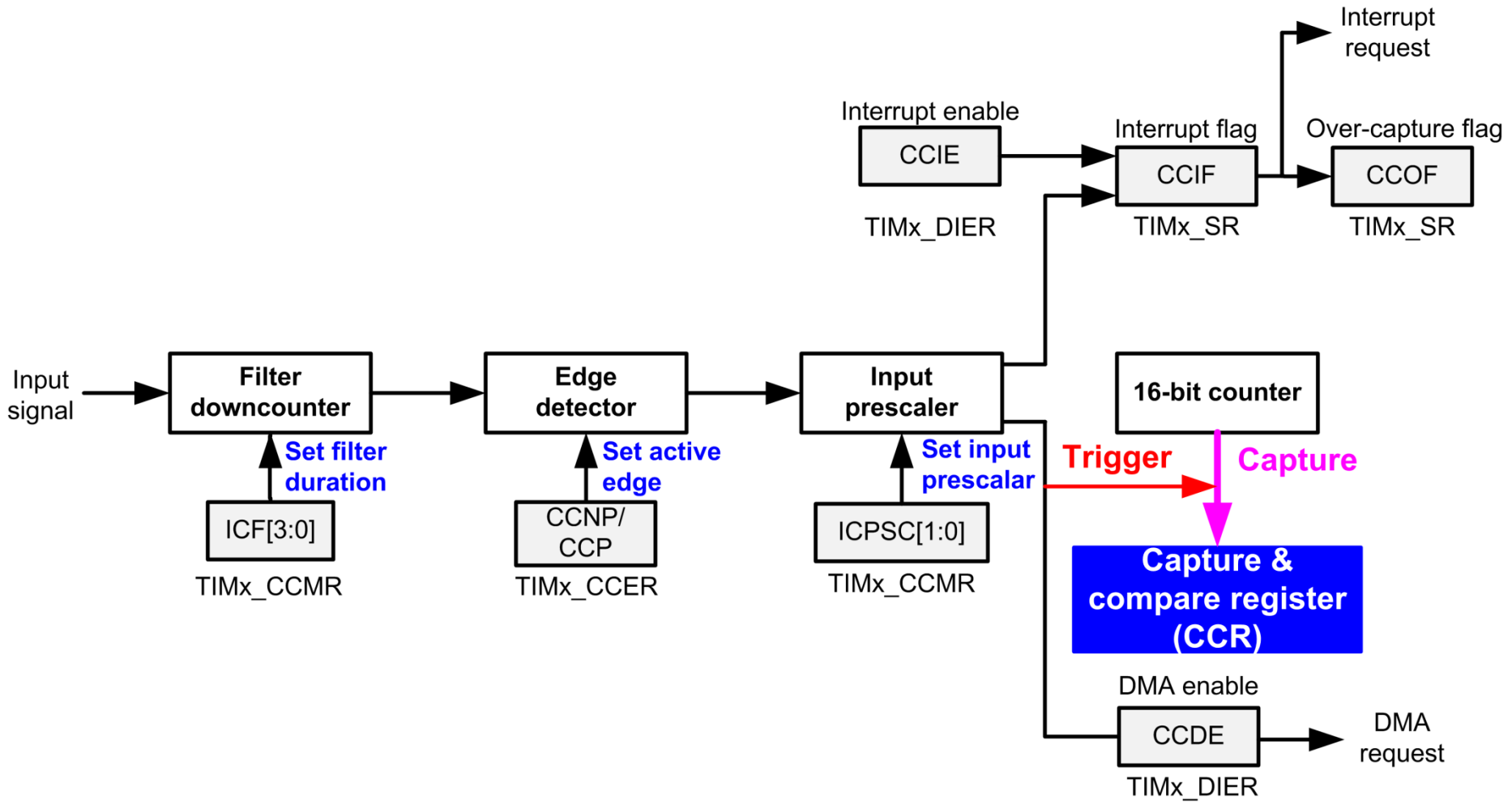
Input Capture



Input Filtering



Input Capture Diagram

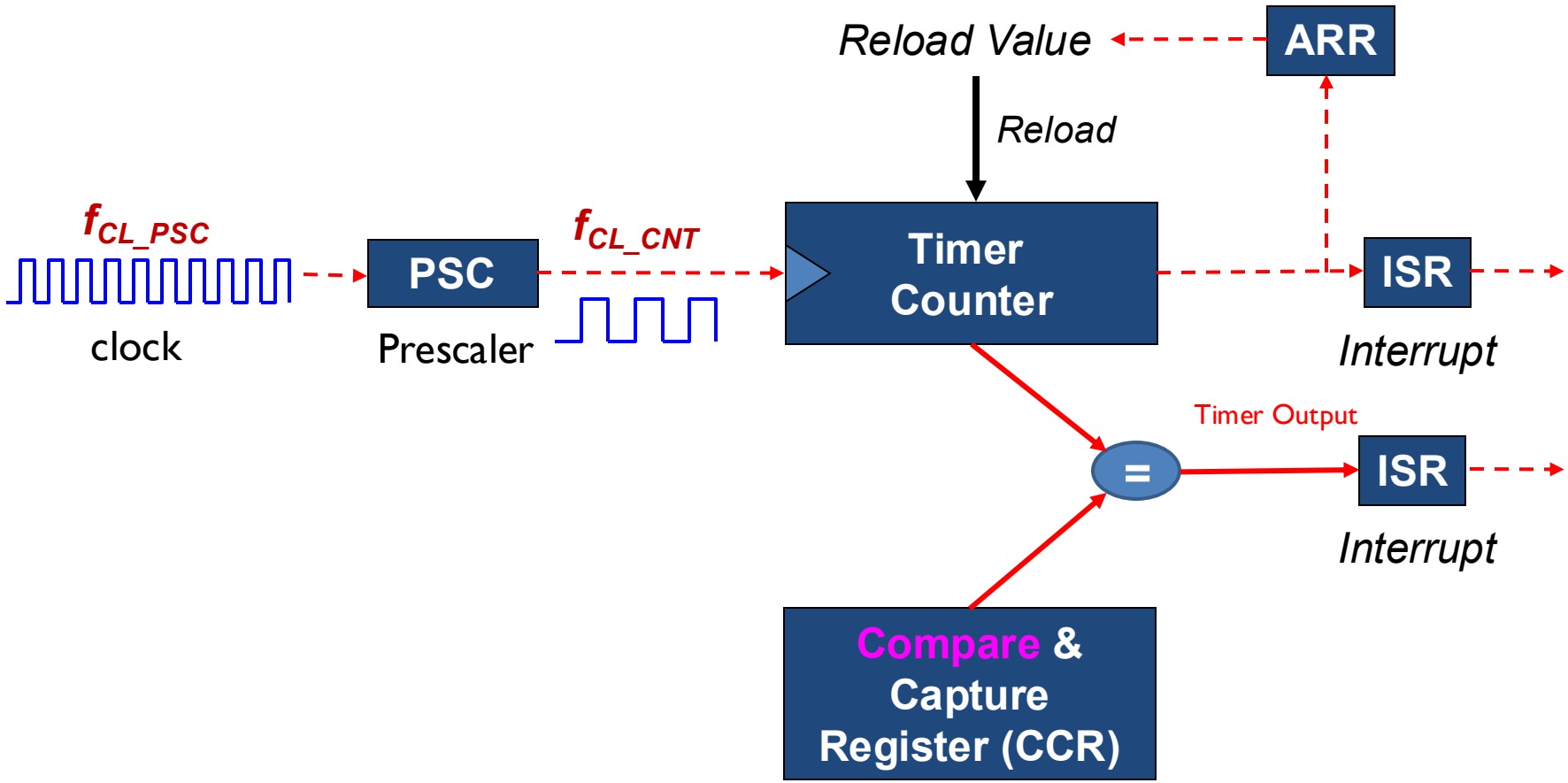


Example # 2: Compare

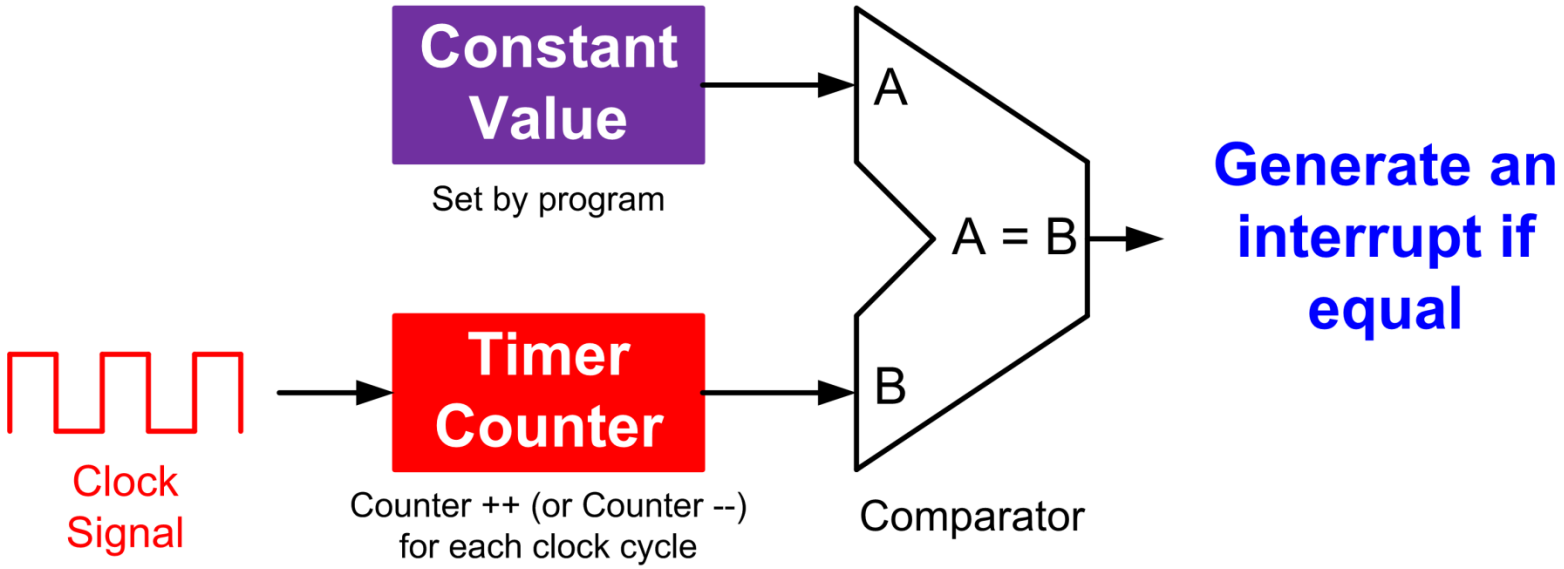
Variable speed Driving a DC motor control

- Motors turn at a speed determined by the voltage applied
  - Doing this in analog can be hard
    - Need to get analog out of our processor
    - Need to amplify signal in a linear way (op-amp?)
  - Generally prefer to switching between “Max” and “Off” quickly
    - Average is good enough.
    - Now don’t need linear amplifier—just “on” and “off”. (transistor)
  - Need a signal with a certain duty cycle and frequency
    - That is % of time high

Timer: Compare

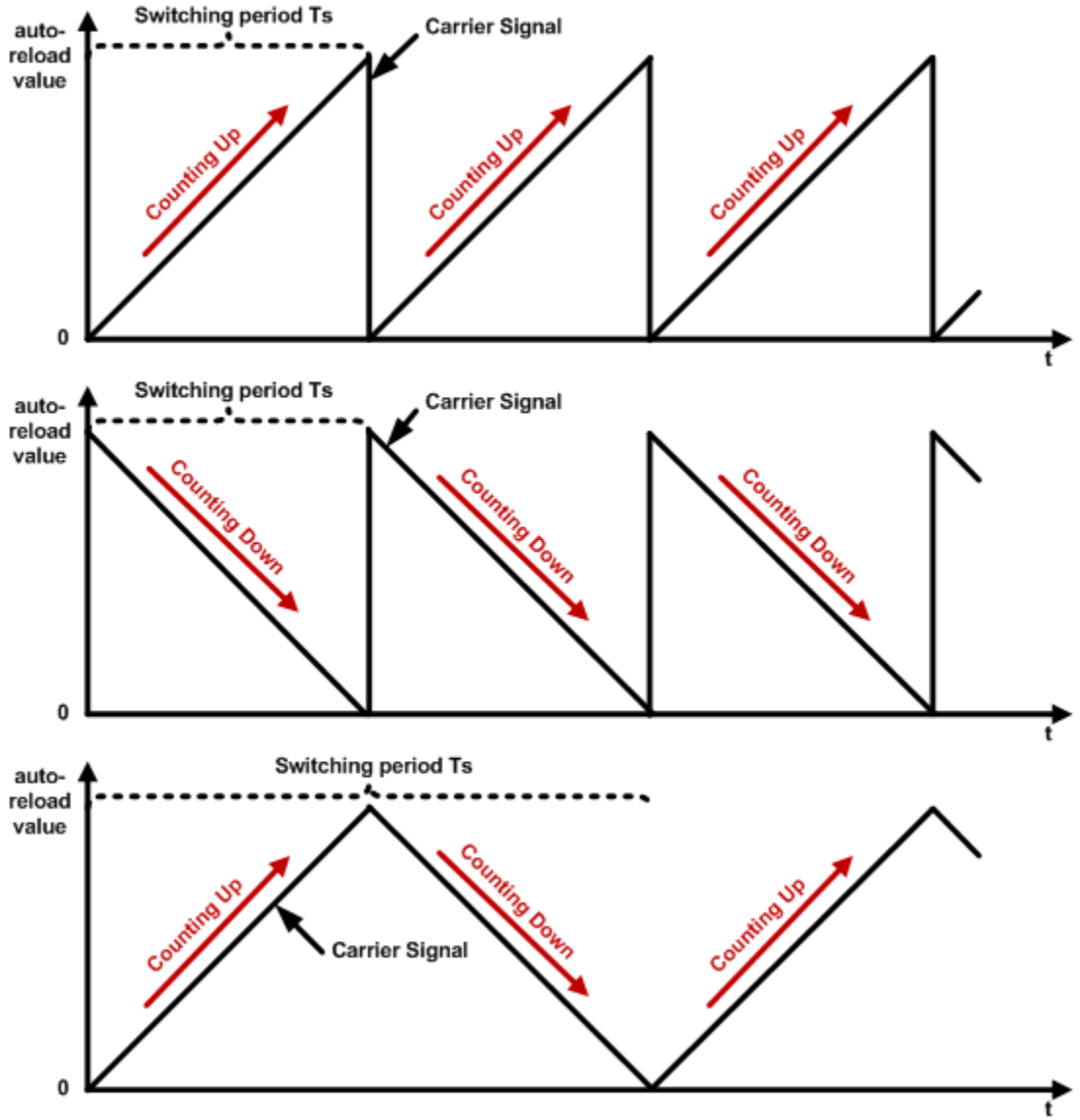


Example: Output Compare

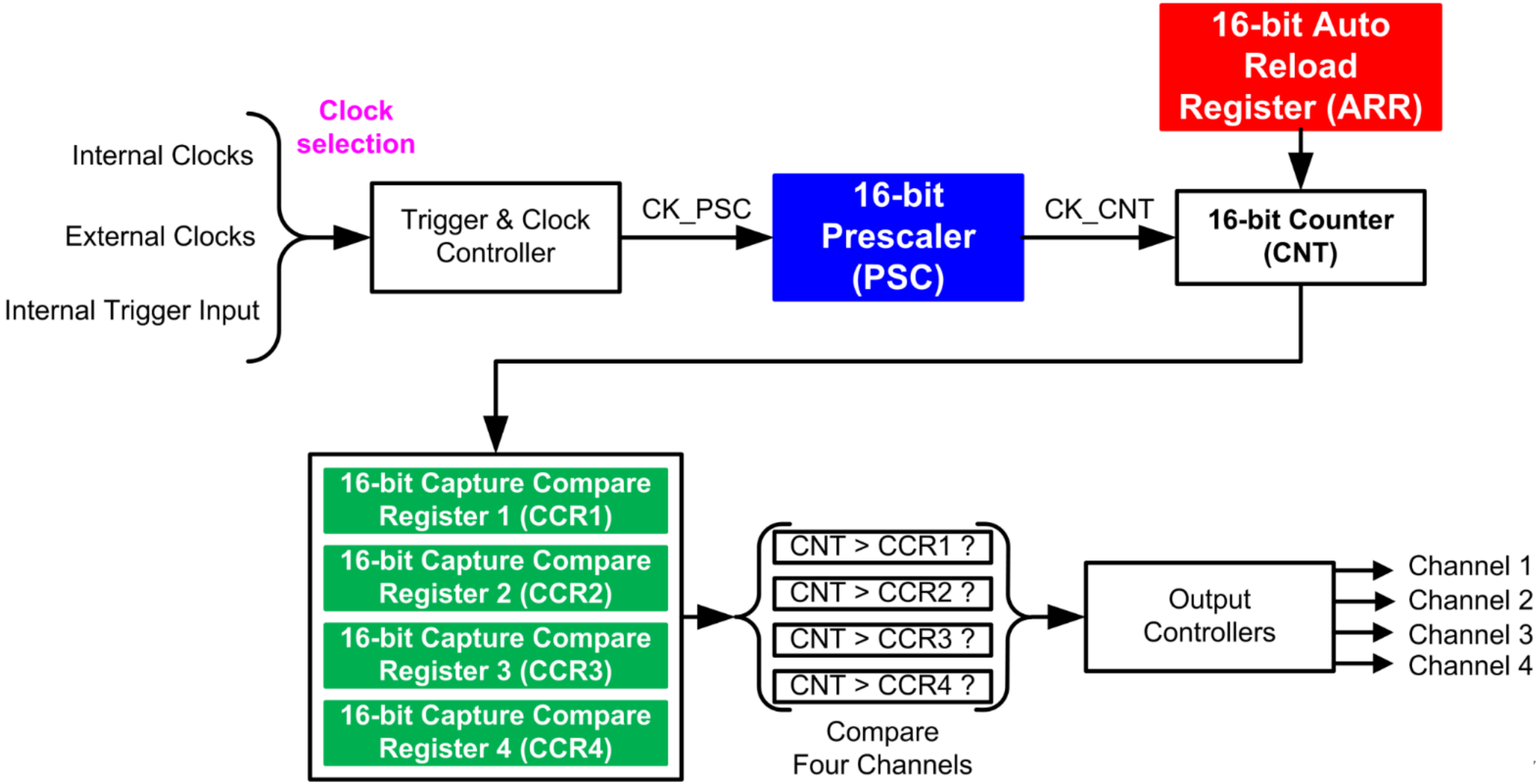


Output Compare Mode (OCM)	Timer Output (OCREF)
000	Frozen
001	High if CNT == CCR
010	Low if CNT == CCR
011	Toggle if CNT == CCR
100	Forced low (always low)
101	Forced high (always high)

Counting up, down, center

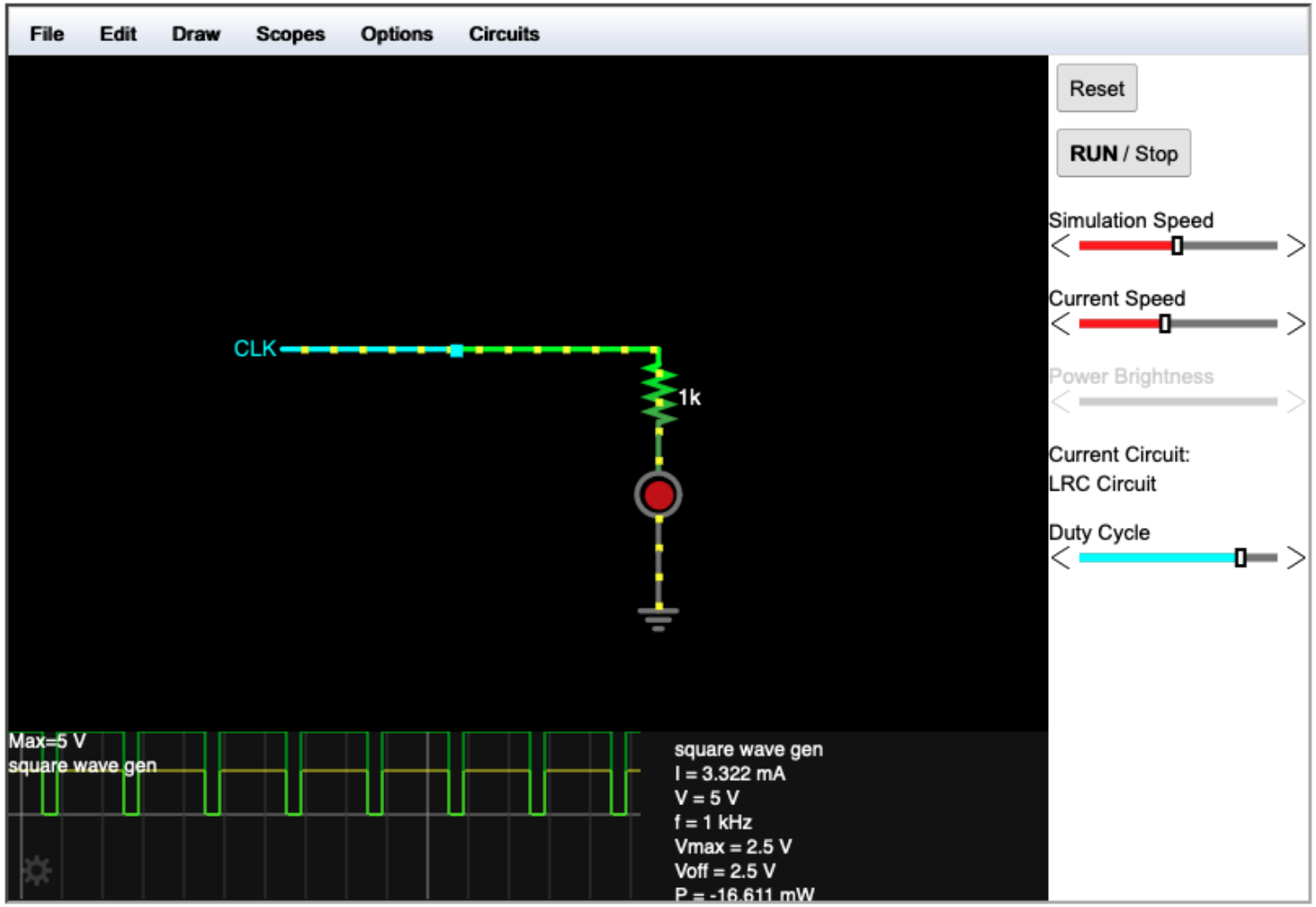


Multi-Channel Outputs



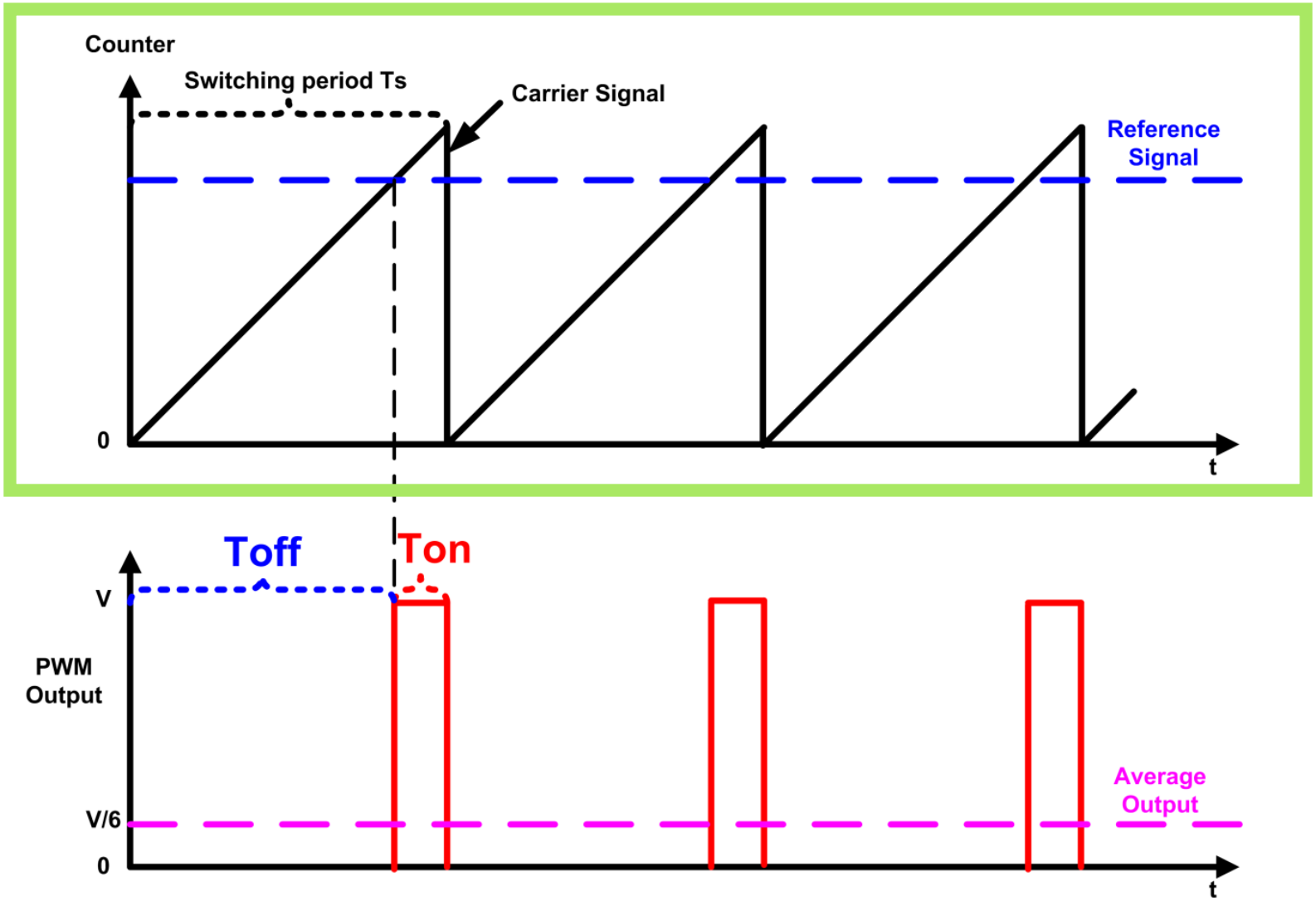
Pulse Width Modulation

Falstad: PWM

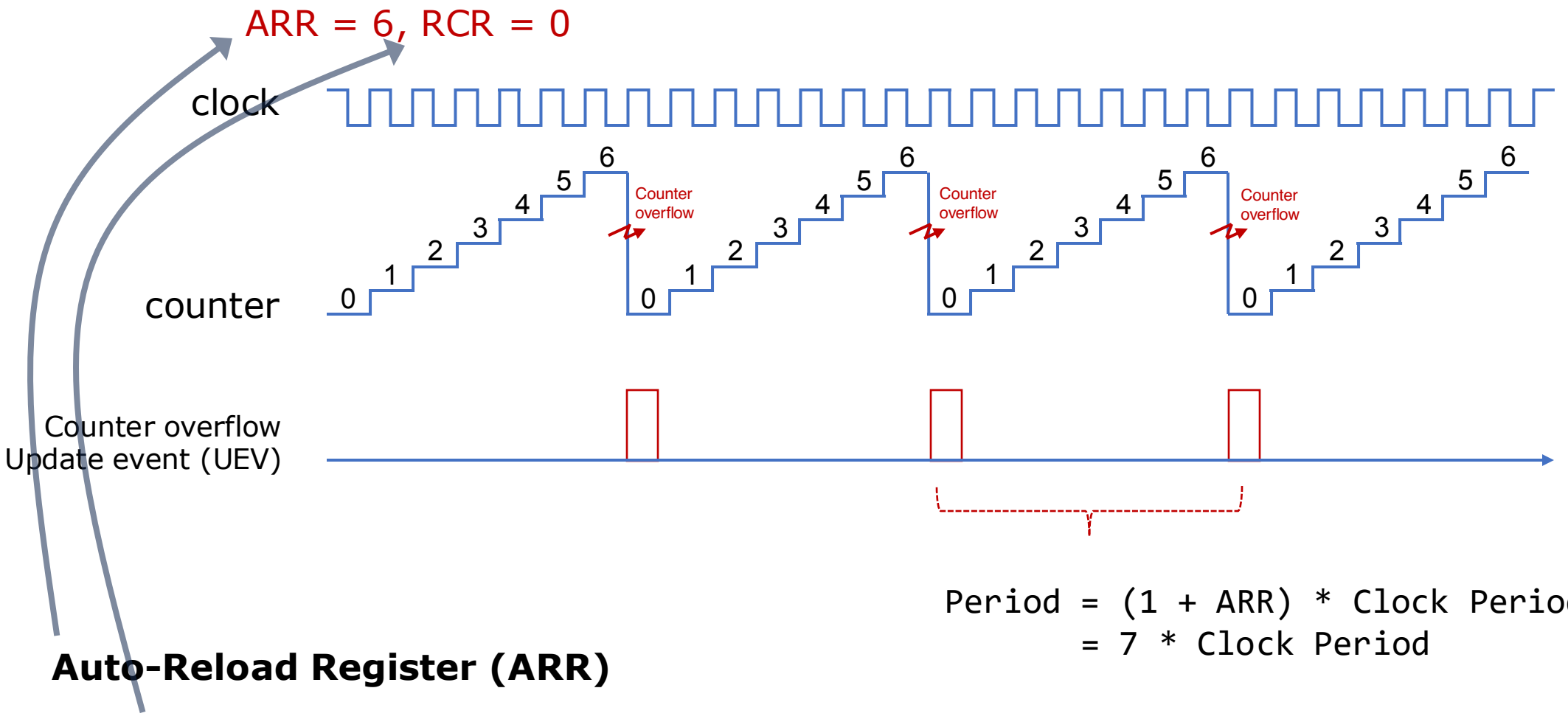


https://tinyurl.com/2ac5j4t5

Pulse Width Modulation Mode



Edge-aligned Mode (Up-counting)




```
Period = (1 + ARR) * Clock Period
       = 7 * Clock Period
```

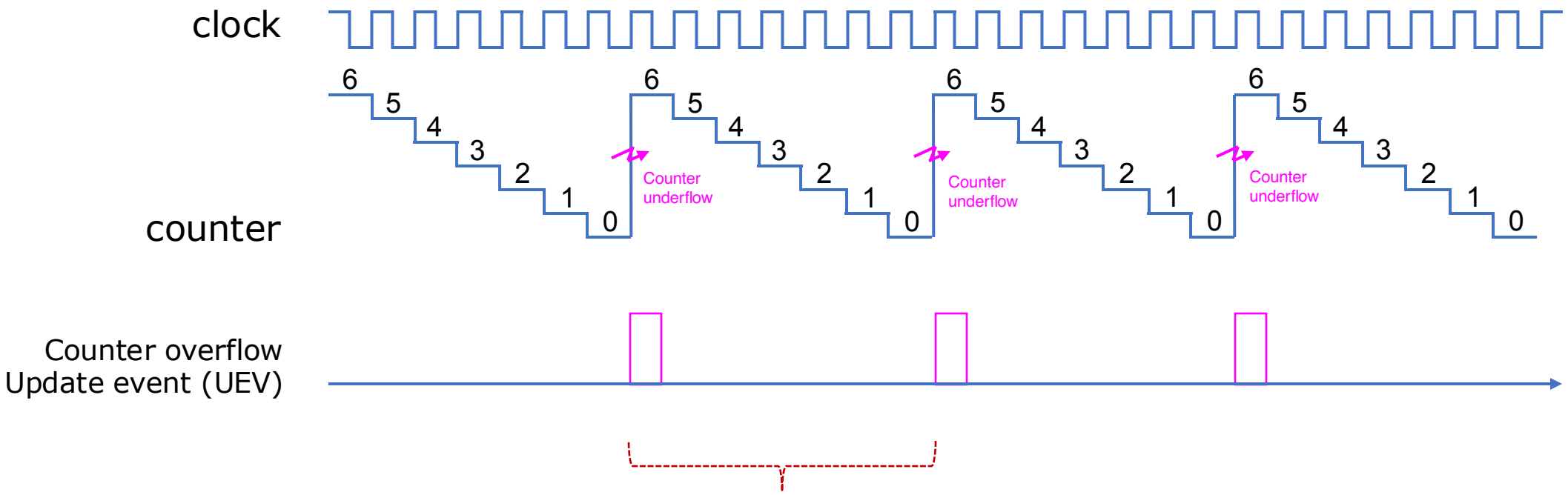
Auto-Reload Register (ARR)

Repetition Counter Register (RCR)

0 = Repeat for ever. Set to a non-zero value, the timer counts the number of occurrences of an update event before generating an interrupt

Edge-aligned Mode (down-counting)

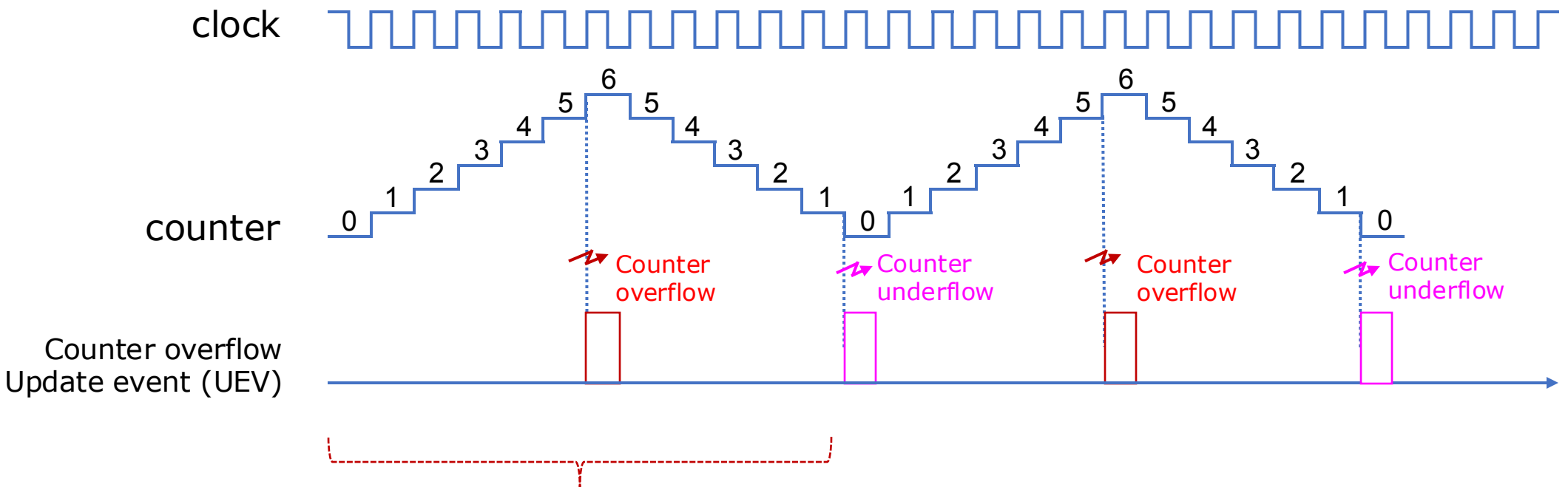
ARR = 6, RCR = 0




```
Period = (1 + ARR) * Clock Period
       = 7 * Clock Period
```

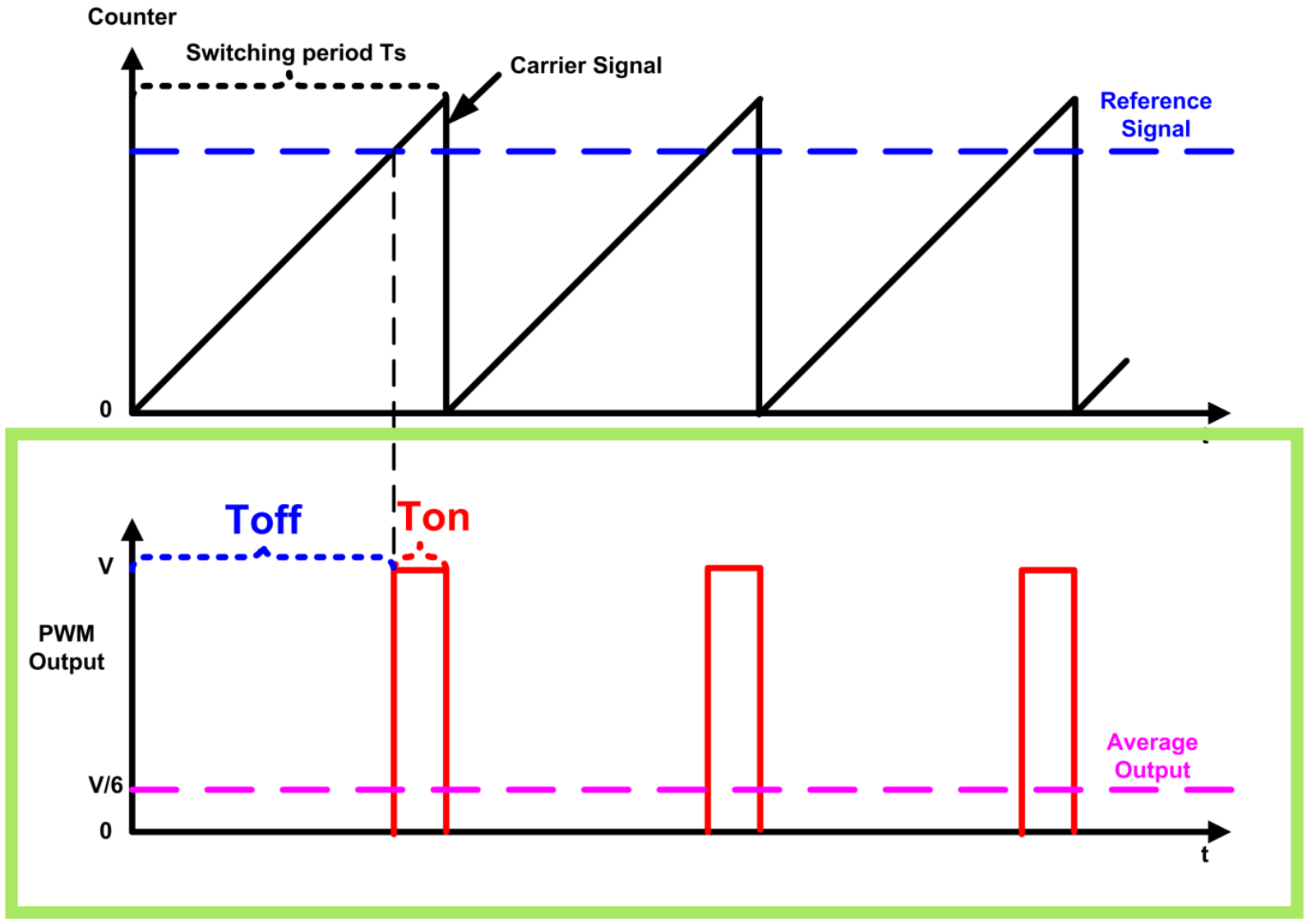
Center-aligned Mode

ARR = 6, RCR = 0




```
Period = 2 * ARR * Clock Period
       = 12 * Clock Period
```

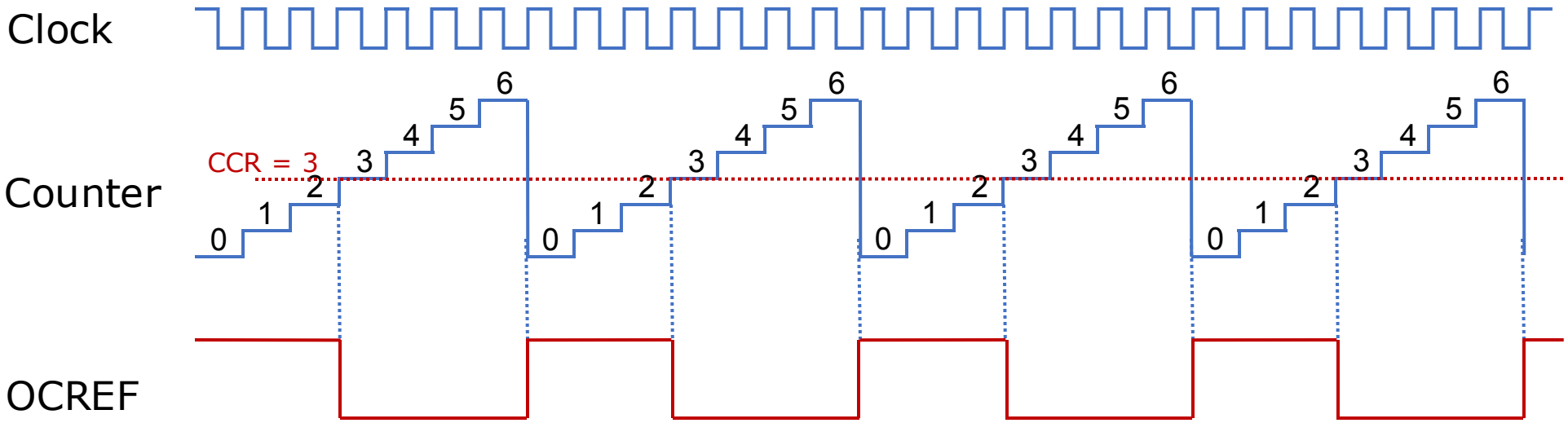
Pulse Width Modulation Mode



PWM Mode 1 (Low-True)

Mode 1
Timer Output = { High if counter < CCR ; Low if counter ≥ CCR }

Upcounting mode, ARR = 6, CCR = 3, RCR = 0



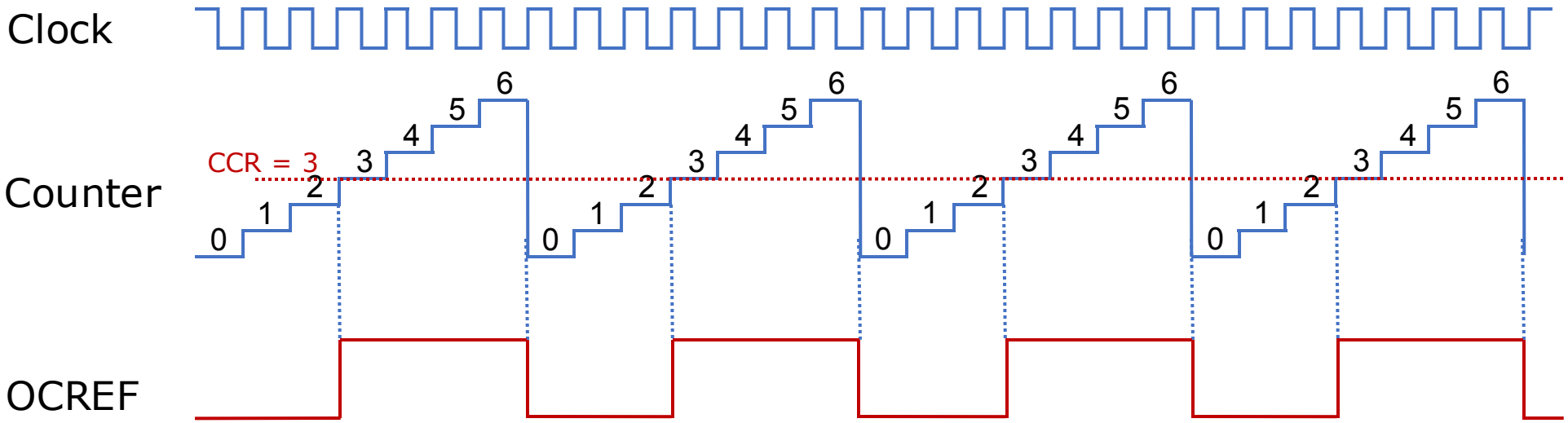
$$\text{Duty Cycle} = \frac{\text{CCR}}{\text{ARR} + 1} = \frac{3}{7}$$

$$\text{Period} = (1 + \text{ARR}) * \text{Clock Period} = 7 * \text{Clock Period}$$

PWM Mode 2 (High-True)

Mode 2
Timer Output = { Low if counter < CCR ; High if counter ≥ CCR }

Upcounting mode, ARR = 6, CCR = 3, RCR = 0



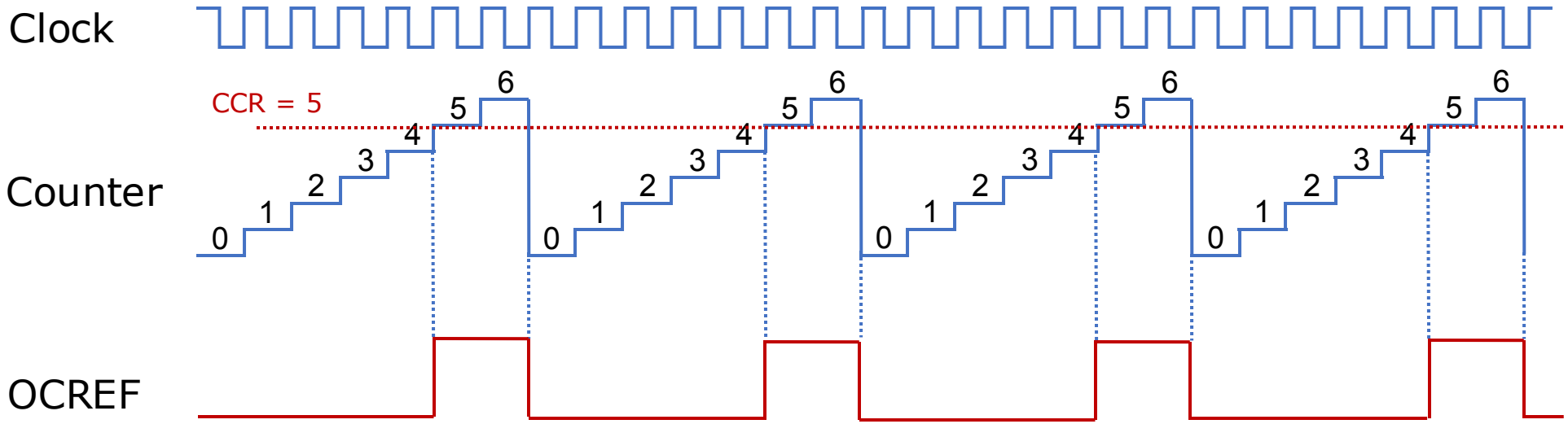
$$\text{Duty Cycle} = 1 - \frac{CCR}{ARR + 1} = \frac{4}{7}$$

$$\text{Period} = (1 + ARR) * \text{Clock Period} = 7 * \text{Clock Period}$$

PWM Mode 2 (High-True)

Mode 2
Timer Output = { Low if counter < CCR ; High if counter ≥ CCR }

Upcounting mode, ARR = 6, CCR = 3, RCR = 0



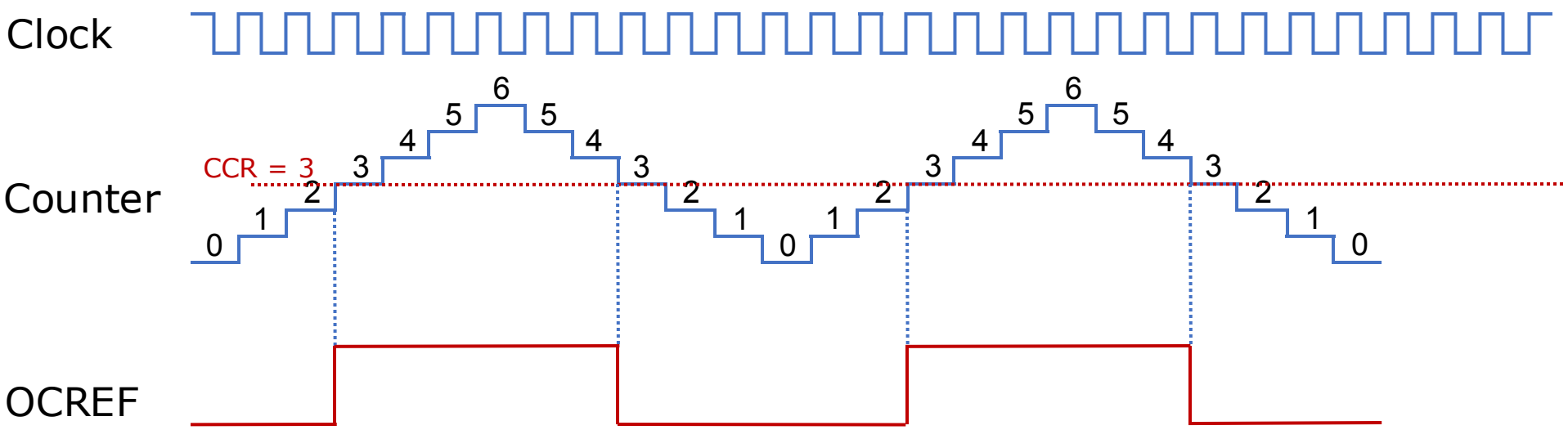
$$\text{Duty Cycle} = 1 - \frac{CCR}{ARR + 1} = \frac{2}{7}$$

$$\text{Period} = (1 + ARR) * \text{Clock Period} = 7 * \text{Clock Period}$$

PWM Mode 2 (High-True)

Mode 2
Timer Output = { Low if counter < CCR ; High if counter ≥ CCR }

Center-aligned mode, ARR = 6, CCR = 3, RCR = 0



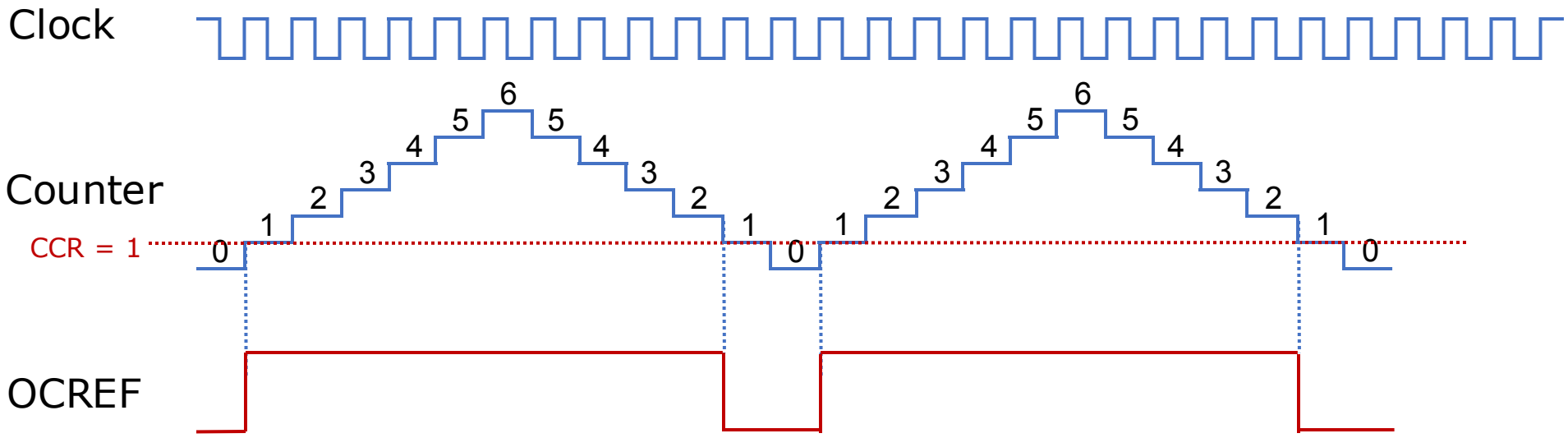
$$\text{Duty Cycle} = 1 - \frac{CCR}{ARR} = \frac{1}{2}$$

$$\text{Period} = 2 * ARR * \text{Clock Period} = 12 * \text{Clock Period}$$

PWM Mode 2 (High-True)

Mode 2
Timer Output = { Low if counter < CCR ; High if counter ≥ CCR }

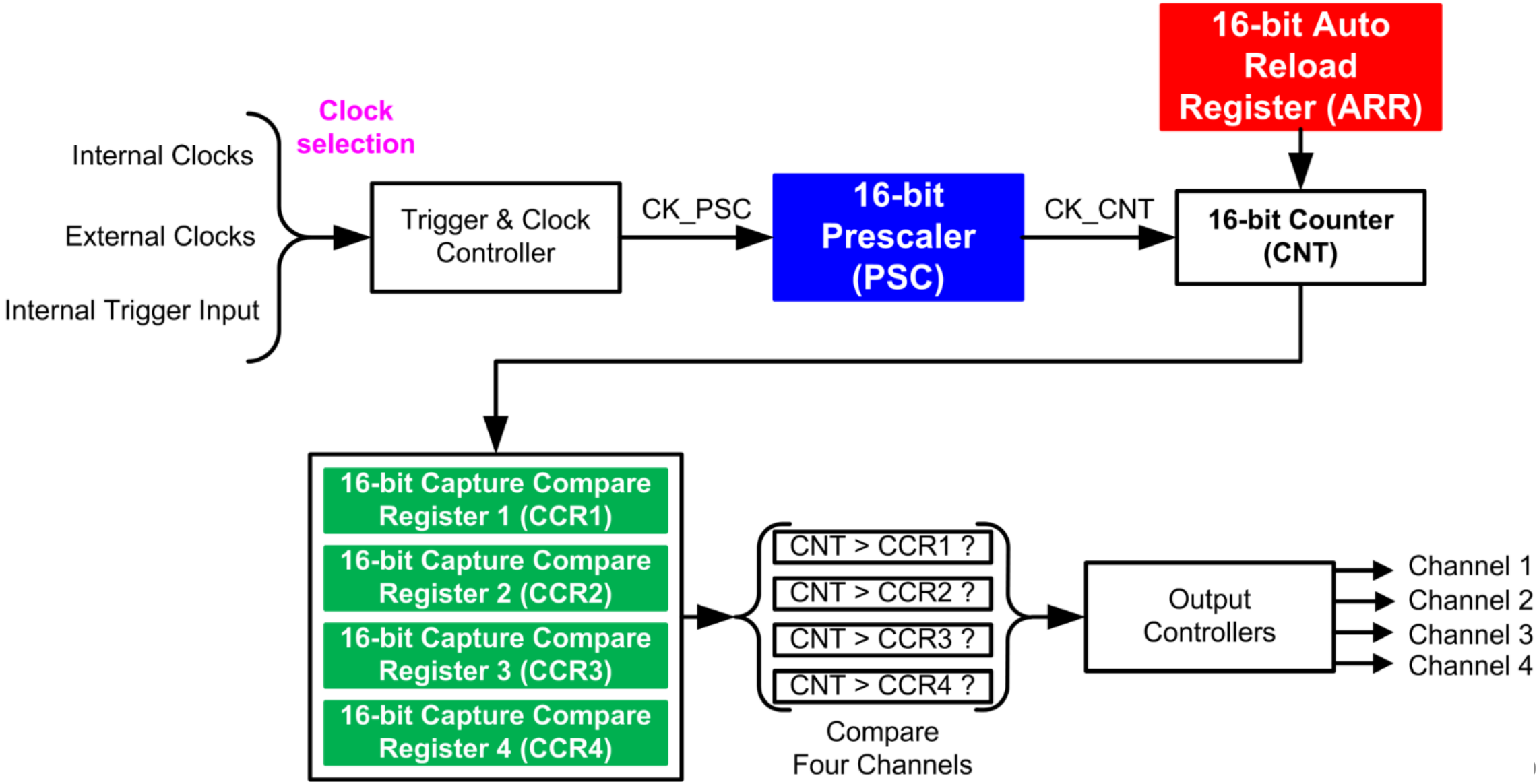
Center-aligned mode, ARR = 6, CCR = 3, RCR = 0



$$\text{Duty Cycle} = 1 - \frac{CCR}{ARR} = \frac{5}{6}$$

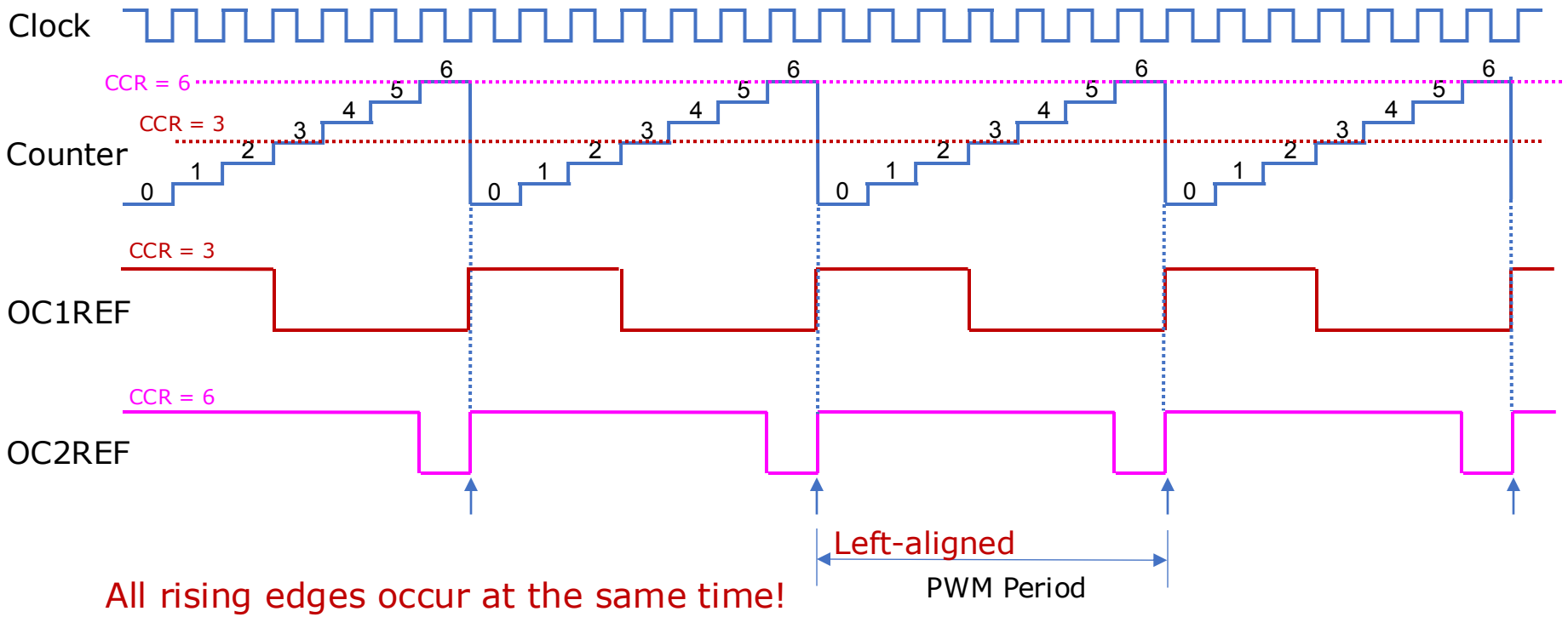
$$\text{Period} = 2 * ARR * \text{Clock Period} = 12 * \text{Clock Period}$$

Multi-PWM outputs from one timer



Up-Counting: Left Edge-aligned

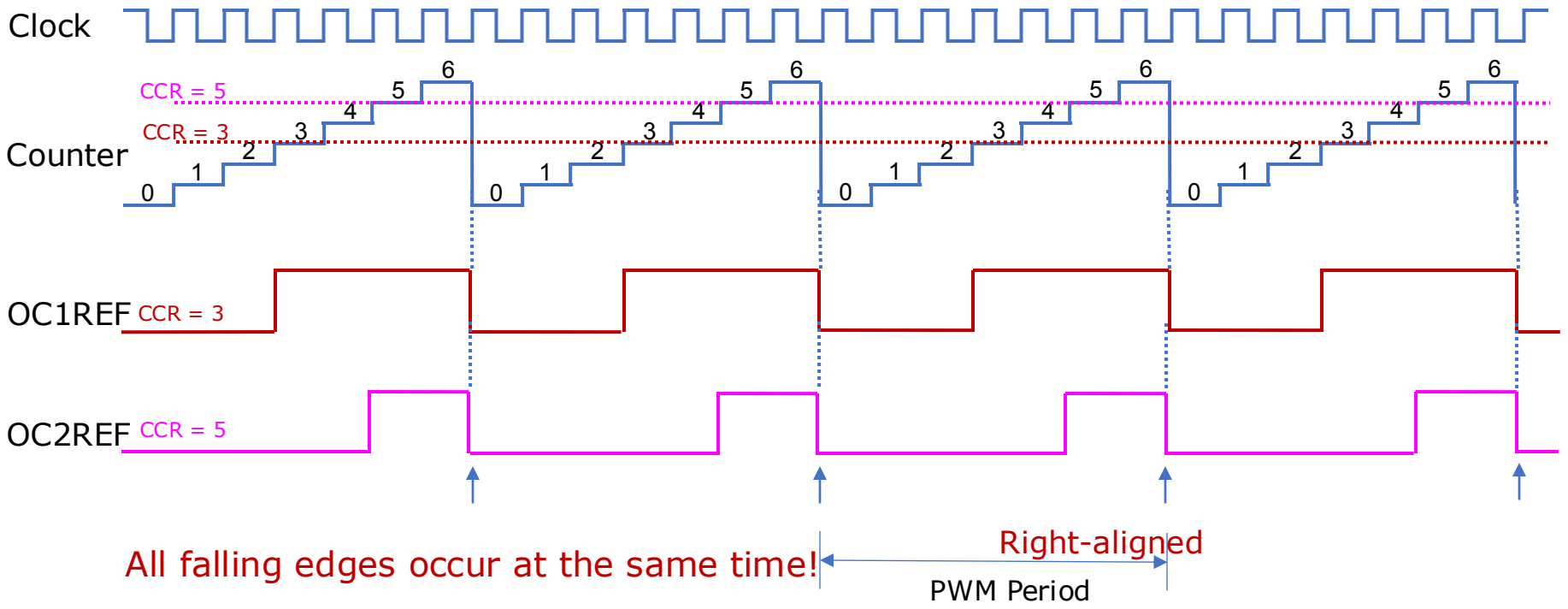
Upcounting mode, ARR = 6, CCR = 3, RCR = 0



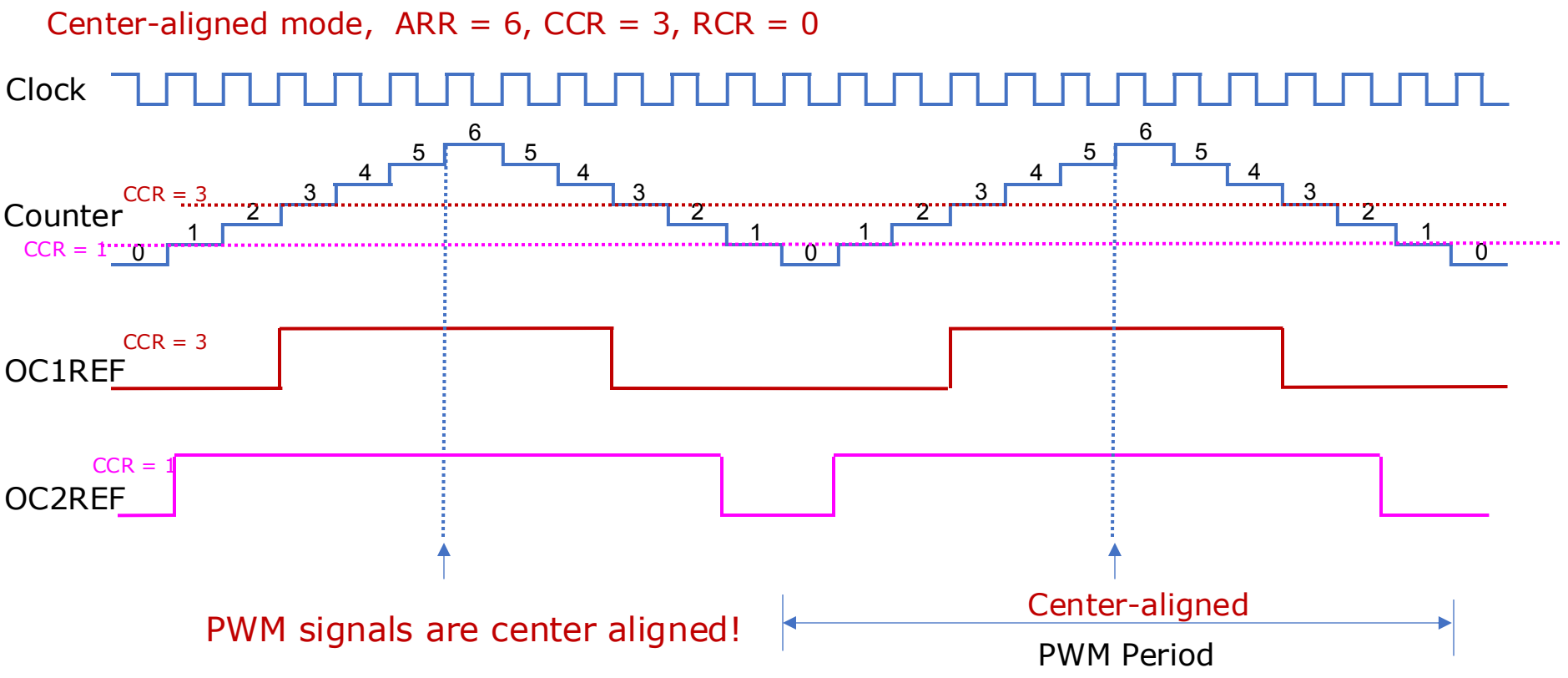
All rising edges occur at the same time!

PWM Mode 2: Right Edge-aligned

Upcounting mode, ARR = 6, CCR = 3, RCR = 0

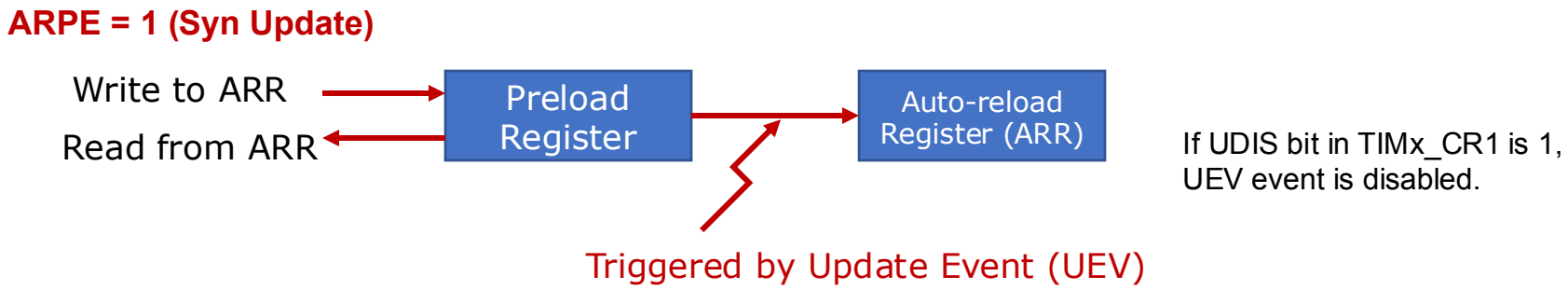


PWM Mode 2: Center Aligned



Auto-Reload Preload Enable (ARPE)

Auto-Reload Preload Enable (ARPE) bit in TIMx_CR1



The auto-reload register (ARR) can be updated synchronously, or asynchronously. If the ARPE bit in the timer CR1 register is 1, ARR will be updated synchronously. An update to ARR will be buffered in a register, named the pre load register. The contents of the pre load register, are transferred into ARR, when the next update event occurs. This update mechanism is synchronous to timer's input clock, and timer's output period. It prevents software from updating the output frequency or period, when the timer is still performing comparison operations.

The devil is in the detail: read the datasheet

Timer output control

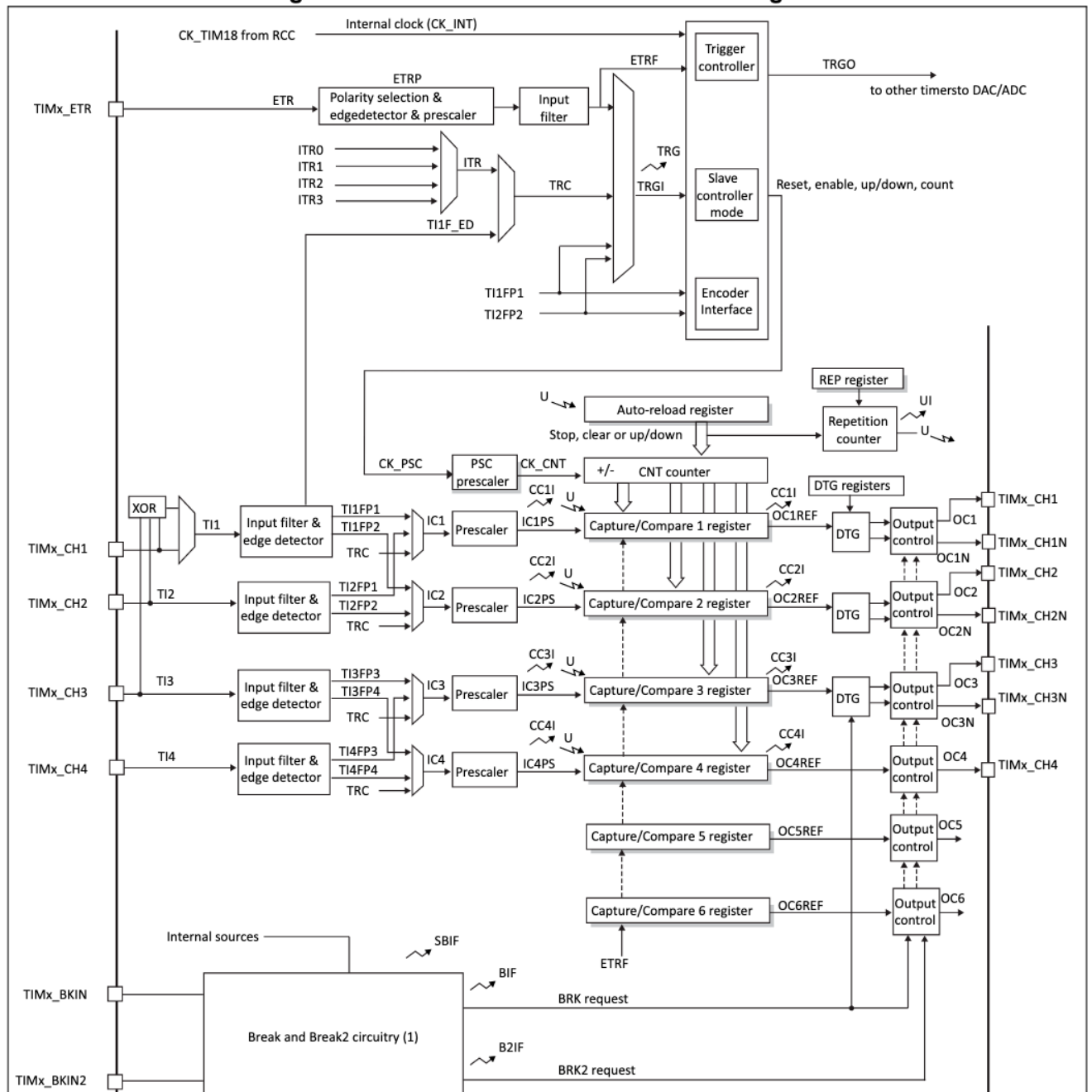
Enable Timer Output

- MOE: Main output enable
- OSSI: Off-state selection for Idle mode
- OSSR: Off-state selection for Run mode
- CCxE: Enable of capture/compare output for channel x
- CCxNE: Enable of capture/compare complementary output for channel x

Table 278. Output control bits for complementary OCx and OCxN channels with break feature

Control bits					Output states(1)	
MOE bit	OSSI bit	OSSR bit	CCxE bit	CCxNE bit	OCx output state	OCxN output state
1	X	X	0	0	Output disabled (not driven by the timer: Hi-Z) OCx=0, OCxN=0	
		0	0	1	Output disabled (not driven by the timer: Hi-Z) OCx=0	OCxREF + Polarity OCxN = OCxREF xor CCxNP
		0	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP	Output Disabled (not driven by the timer: Hi-Z) OCxN=0
		X	1	1	OCREF + Polarity + dead-time	Complementary to OCREF (not OCREF) + Polarity + dead-time
		1	0	1	Off-State (output enabled with inactive state) OCx=CCxP	OCxREF + Polarity OCxN = OCxREF x or CCxNP
		1	1	0	OCxREF + Polarity OCx=OCxREF xor CCxP	Off-State (output enabled with inactive state) OCxN=CCxNP
0	0	X	X	X	Output disabled (not driven by the timer: Hi-Z).	
	1		0	0		
			0	1	Off-State (output enabled with inactive state) Asynchronously: OCx=CCxP, OCxN=CCxNP (if BRK or BRK2 is triggered).	
			1	0		
			1	1	Then (this is valid only if BRK is triggered), if the clock is present: OCx=OISx and OCxN=OISxN after a dead-time, assuming that OISx and OISxN do not correspond to OCX and OCxN both in active state (may cause a short circuit when driving switches in half-bridge configuration). **Note**: BRK2 can only be used if OSSI = OSSR = 1.	

Figure 283. Advanced-control timer block diagram

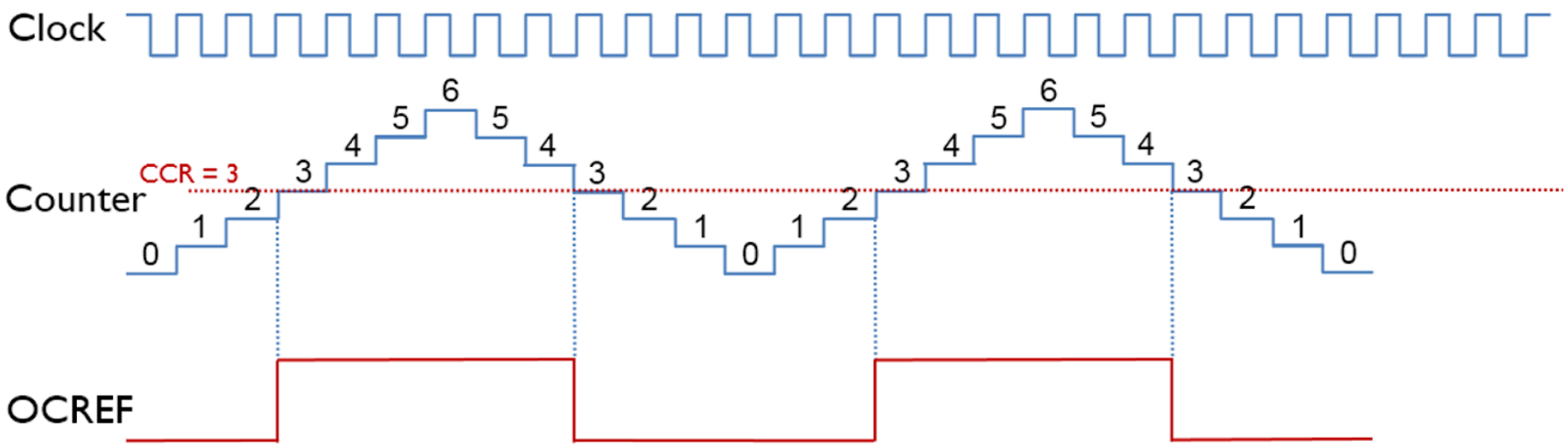


Lecture 9 - Worksheet

PWM Timer Setup

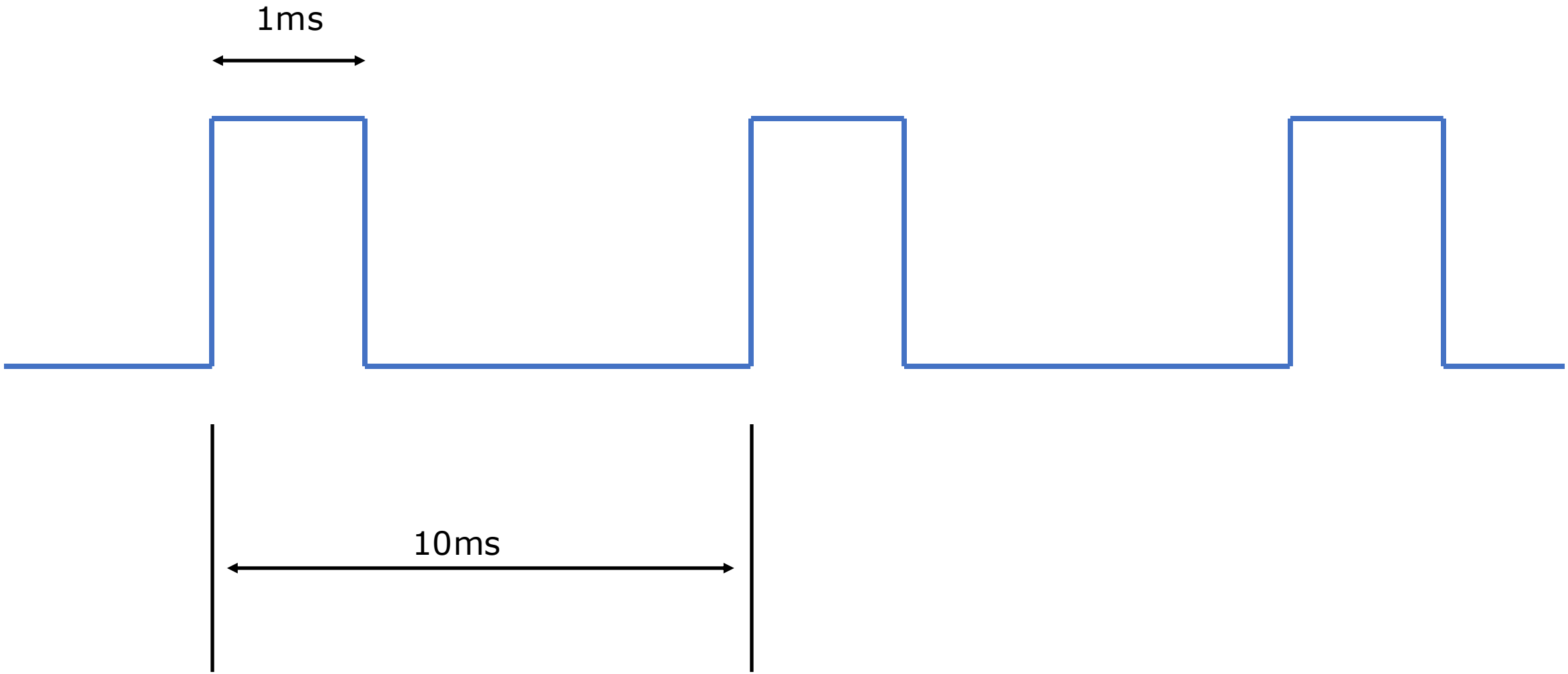
Given the following assumptions, find the timer's PSC, ARR and CCR to achieve a PWM output with a duty cycle of 10% and a period of 10 *ms*.

- 16 MHz clock is selected to drive the timer.
- The timer is set up as repeatedly up-counting up and down-counting, similar to the figure shown below.
- PWM Mode 2 is selected. In this mode, the timer output is high if the counter is larger than or equal to CCR; otherwise, the output is low.



Lecture 9 - Worksheet

$$f_{Clock\_Out} = \frac{f_{Clock\_in}}{PSC + 1}$$



Lecture 9 - Worksheet

Base Equations:

Timer Clock Frequency = Input Clock Frequency / (1 + PSC)
PWM Period = 2 * ARR * Timer Clock Period (up then down count mode)
PWM Duty Cycle = 1 – CCR/ARR

Solution:

Let’s make the clock period something easy = say 1MHz

Timer clock frequency = 16 MHz / (1 + 15) = 1 MHz
PSC = 15

ARR = PWM Period / (2 * Timer Clock Period)
= PWM Clock Period * Timer Clock Frequency / 2
= 10 ms * 1 MHz / 2
= 5000

CCR = (1 – Duty Cycle) * ARR
= (1 - 10%) * 5000
= 4500

Recap

- Oscillators
  - Low frequency crystals
  - High frequency resonates
  - Internal oscillators
  - PLL
- Clock Distribution & Prescalers
- Timers
  - Capture
  - Compare
- PWM Generation & timer modes

-end

Acknowledgement

I an effort to make sure credit is given where it is due. These lecture and slides have been adapted and/or influenced by the following people:

Dr. Mark Brehob – University of Michigan

Dr. Ronald Dreslinski – University of Michigan

Dr. Yifeng Zhu – University of Maine